Inhalt

Dank

An dieser Stelle möchte ich den Personen danken, die mich dabei begleitet haben, einen Teil der sich mir im Zusammenhang mit freier Software stellenden Fragen zu beantworten. Nennen möchte ich Prof. Wolfgang Krohn, dem ich nicht nur mein grundlegendes Verständnis von Technik verdanke, sondern der meinem Projekt immer wohlwollend und fördernd gegenüber stand; Prof. Alfons Bora, der durch die Klärung von methodischen Fragen, den Platz den er meiner Untersuchung in der Methodenwerkstatt einräumte und den Hinweis darauf, die Entstehung von freien Softwareentwicklungsprojekten nicht einfach unhinterfragt vorauszusetzen, die Arbeit stark geprägt hat; Prof. Peter Weingart, der mir gezeigt hat, was einen guten Chef ausmacht; Andreas Pettenkofer, der in unnachahmlicher Weise produktiv fragen und insistieren kann und der mir so zu unzähligen Einsichten verholfen hat; meine Interviewpartner, insbesondere die Intevation GmbH und die Free Software Foundation Europe (FSF), die meine Fragen und Nachfragen geduldig beantwortet und einem Unwissenden zumindest zu ein wenig Verständnis ihrer Tätigkeit verholfen haben; und meine Eltern, die bei meinem Ansinnen, in einer von außen vermutlich esoterisch anmutenden Wissenschaft wie der Soziologie zu promovieren, nicht nervös geworden sind. Mein Dank gilt auch der Deutschen Forschungsgemeinschaft, die das Projekt über drei Jahre hinweg im Rahmen des Graduiertenkollegs *Genese, Strukturen und Folgen von Wissenschaft und Technik* mit einem Promotionsstipendiums gefördert hat und dem Graduiertenkolleg *Auf dem Weg in die Wissensgesellschaft* am Institut für Wissenschafts- und Technikforschung (IWT), das die vorliegende Publikation mit einem Druckkostenzuschuß unterstützt hat. – Niels C. Taubert

Einleitung

Erfolgsgeschichten gab es im IT- und Softwarebereich in den vergangenen Jahren viele – wenn auch mit durchaus unterschiedlichem Ausgang. Kinder des Internet-Booms traten in großer Zahl und mit viel Getöse auf, zogen nicht nur Risikokapital, sondern in den 1990er Jahren auch das Geld von Kleinanlegern an, um dann nach mehr oder minder lang anhaltendem Höhenflug der Aktienkurse kurz von dem Jahrtausendwechsel herbe Abstürze hinzulegen. Aus diesem Boom erwachsende Unternehmen waren dann doch selten, häufig endete der Erfolg in Katerstimmung: Dem ‚rise' folgte – wie so häufig – ein noch prompterer ‚fall'. Verläßlicher Lieferant guter Nachrichten im Bereich der Informations- und Kommunikationstechnologien bildete während der gesamten Zeit freie Software: Von der Verbreitung und Popularisierung des Entwicklungsmodells Anfang/Mitte der 1990er Jahre bis zum heutigen Zeitpunkt läßt sich eine Geschichte des Erfolgs schreiben, die derartige Einbrüche nicht kennt. Die Zahl freier Programme und deren Verbreitung steigt kontinuierlich.

Zum Beispiel Linux: Als prominentester Vertreter dieses Softwaretypus ist das Betriebssystem mittlerweile zu einer etablierten Erscheinung auf Rechnerarbeitsplätzen geworden. Gestartet als System von Programmierern für Programmierer, hat es stark an Benutzerfreundlichkeit gewonnen und kann auch ohne die Kenntnis kryptischer Befehle von jedermann bedient werden. So gereift, hat es seinen Weg vom Exotensystem für Computerfreaks in die Unternehmen,[1] die Bundestagsver-

1 Siehe hierzu beispielsweise die Liste der Referenzkunden des Linux-Distributors Suse unter http://www.suse.de/de/company/customer_refer-ences/index.html. Alle in diesem Buch angegebenen Internetseiten wurden, sofern nicht anders angegeben, zuletzt im Oktober 2004 gesehen.

waltung,[2] die Stadtverwaltungen,[3] öffentliche Einrichtungen,[4] das Militär[5] und auch den heimischen PC gefunden.

Zum Beispiel Apache: Dieser Webserver bietet von seiner Verbreitung her ein noch größeres Erfolgsbeispiel. Ebenfalls als freies Softwareentwicklungsprojekt Anfang 1995 in der Version 1.0 veröffentlicht, gilt er seit spätestens 1996 als der am weitesten verbreitete Webserver, vom dem nach jüngsten Umfragen mehr als 62 Prozent der Websites im WWW gehostet werden.[6] Die entsprechenden Konkurrenzprogramme der Unternehmen Microsoft, Zeus und Sun folgen weit abgeschlagen.

Zum Beispiel Firefox: Jüngstes Beispiel bildet dieser Webbrowser, ein Ableger von Mozilla. Mitte September 2004 wurde die Version 1.0 der Öffentlichkeit vorgestellt und zum Download freigeschaltet. 300.000 Downloads in den ersten 24 Stunden künden von einem riesigen Interesse an dem Programm und lassen die Entwickler darauf hoffen, innerhalb von einem Monat von Null auf zehn Millionen Nutzern zu kommen.[7]

Überwältigend ist neben der schnellen Verbreitung von freier Software auch der Ruf, der diesem Programmtypus vorauseilt: Kurz nachdem das Phänomen öffentlich wahrgenommen wurde, war die Berichterstattung euphorisch und ist heute immer noch positiv-wohlwollend – kritische Anmerkungen und Kommentare sind nach wie vor spärlich. Freie Software gilt als die freundliche und begrüßenswerte Alternative zu dem sonstigen Einerlei, das durch einen weitgehend negativ etikettierten amerikanischen Softwarekonzern geschaffen wurde. In diesem Spiel David gegen Goliath sind die Sympathien klar verteilt: David gilt hier als die Partei, für die das Herz schlägt, auch wenn vorerst noch die Software von Goliath genutzt wird.

Diese positiven Zuschreibungen setzen sich auf technischer Ebene fort: Freie Software gilt als stabiler, verschont den leidgeplagten Nutzer

---

2 Vgl. hierzu den Bericht über die Entscheidung des Ältestenrats in der Neuen Zürcher Zeitung vom 14.03.04. Als Online-Dokument: http://www.nzz.ch/2002/03/08/em/page-article80M1P.html.

3 So zum Beispiel die Stadtverwaltung München. Siehe den Spiegel Online Bericht vom 02.06.04. Als Online-Dokument: http://www.spiegel.de/spiegel/0,1518,251077,00.html.

4 Beispiel hierfür bilden die Terminals der Universitätsbibliothek Bielefeld.

5 Der vom Pentagon bei IBM in Auftrag gegebene Supercomputer soll aus 1186 IBM-Rechnern zusammengeschaltet werden, die US Navy mit aktuellen Wettervorhersagen versorgen und ein Linux-Betriebssystem besitzen. Siehe den Bericht vom 04.08.04 in Spiegel Online unter http://www.spiegel.de/netzwelt/netzkultur/0,1518,311542,00.html.

6 Vgl. die Netcraft-Umfragen vom Dezember 2002. Als Online-Dokument: http://www.netcraft.com/Survey/index-200212.html.

7 Vgl. Spiegel Online vom 16.09.04. Als Online-Dokument: http://www.spiegel.de/netzwelt/technologie0,1518,druck-318189,00.html.

von Systemabstürzen und verhält sich damit ressourcenschonend – zumindest was die Nerven der Anwender angeht. Freie Software gilt als weniger betroffen von Viren, Würmern, Trojanern und Spybots, also all jenen Begleiterscheinungen der vernetzten Gesellschaft, die dem Anwender das Fürchten lehren und ihn veranlassen, regelmäßig lästige Sicherheitsupdates durchzuführen, auch wenn er weiß, daß ihm diese eine nur äußerst kurzzeitige Atempause verschaffen – bis eben die nächste kritische Sicherheitslücke entdeckt wird, deren Größe mittlerweile die eines Scheunentors annehmen kann.[8] Selten war der Ruf eines bestimmten Typus von Technik so gut, wie es bei freier Software der Fall ist.

Worauf gründet sich nun dieser Erfolg, der sich in Ruf wie Verbreitung zeigt? Liegt es daran, daß freie Software in aller Regel kostenlos erhältlich ist, der Anwender sie also nutzen kann, ohne dafür zu bezahlen – frei nach dem Motto: ‚Einem geschenkten Gaul schaut man nicht ins Maul'? Liegt es an der tatsächlichen Überlegenheit der Qualität des Programms, die aus dem Modell resultiert, mit dem es entwickelt wird und das dem elektronischen Wunderland direkt entsprungen zu sein scheint? Oftmals als chaotisch oder als anarchisch beschrieben, kann sich jedermann an der Softwareentwicklung beteiligen, das Programm nach seinen Wünschen modifizieren, was in der Summe dann wie von Zauberhand geleitet oftmals zu einem sehr brauchbaren Ergebnis führt.

Einen Teil des Erfolgs erklären wir in dieser Arbeit, indem wir der Frage nachgehen, wie freie Software innerhalb von offenen Programmierprojekten entwickelt wird. Das zumindest im Fall von erfolgreichen Projekten gelingende Zusammenwirken in Richtung eines Projektziels findet dabei unter Rahmenbedingungen statt, die auf den ersten Blick für das Entstehen von Handlungskoordination wenig günstig erscheinen: Freie Softwareentwicklungsprojekte sind öffentlich – jeder kann also mitmachen, aber genauso schnell wieder abspringen. Die Entwickler sind über den Globus verteilt und koordinieren ihre Arbeit ausschließlich über das Internet. Es findet sich bei näherer Betrachtung keine Organisation, welche die Koordinationsleistung in freien Softwareentwicklungsprojekten erbringt und die notwendigen Ressourcen für die Entwicklung bereitstellt. Die Existenz freier Software wird bei näherer Betrachtung zunehmend erklärungsbedürftig.

Die Studie gliedert sich wie folgt: In einem ersten Teil legen wir die Grundlagen für die Untersuchung. Am Anfang steht hier die Klärung der Begriffe ‚freie Software' und ‚freie Softwareprojekte', mit denen wir den hier interessierenden Gegenstand bestimmen. Hieran anschließend

8 Vgl. Spiegel Online vom 08.07.04. Als Online-Dokument: http://www.spiegel.de/netzwelt/technologie/0,1518,druck-307447,00.html.

werden wir im zweiten Kapitel die Fragestellung entwickeln. Abgeschlossen wird dieser Grundlagenteil mit der Darstellung des theoretischen Rahmens der Untersuchung, dem Konzept der Innovationsnetzwerke, mit dem Kooperationsprozesse zwischen heterogenen Akteuren im Bereich der Technikentwicklung analysiert werden.

Im Fokus des zweiten Teils steht dann die Frage nach den Ursachen für die Entstehung des Netzwerks freier Softwareentwicklung, das sich in die für das Feld typischen freien Softwareentwicklungsprojekte gliedert. Wie wir sehen werden, ist diese Struktur zwei Innovationen geschuldet: zum einen der vertraglichen Innovation freier Softwarelizenzen, zum anderen der organisatorischen Innovation der offenen Programmierprojekte. Im ersten Kapitel des zweiten Teils werden wir diese spezielle Form der Lizenz diskutieren und zeigen, daß diese sich reflexiv auf das Urheberrecht bezieht: Die dort monopolisierten Nutzungsrechte am Programm zugunsten des Werkschöpfers werden genutzt, um sie zu generalisieren: *All rights reversed*. Daran anschließend analysieren wir die zweite Innovation der offenen Programmierprojekte. Bezugspunkt dieser Innovation bildet das Software Engineering, eine Teildisziplin der Informatik, die ein Set von normativen Vorgehensmodellen bereithält, in denen sich die leitenden Vorstellungen über die Organisation von Softwareentwicklungsprozessen finden. Mit der Analyse der Diskussionen, die zum ersten offenen Programmierprojekt geführt haben, werden wir zeigen, daß sich die Innovation weitgehend negativ auf das Software Engineering bezieht und an die Stelle einer hierarchischen Projektsteuerung die Idee einer selbstorganisierten Projektabwicklung setzt.

Nachdem wir die Ursachen für das Entstehen dieser sozialen Struktur analysiert haben, werden wir uns im dritten Teil – Innenansichten offener Programmierprojekte – der Analyse der sozialen Mechanismen zuwenden, die für eine Handlungskoordination sorgen. Zunächst werden wir das Fallbeispiel KMail vorstellen, um daran anschließend in Form eines Exkurses die Gründe der Akteure für eine Beteiligung an der freien Softwareentwicklung zu analysieren. Im dritten Kapitel dieses Teils steht dann der Projektbeginn und die dabei von dem oder den Gründer(n) zu bewältigenden Aufgaben sowie eine im Bereich der freien Softwareentwicklung typischerweise gefundene Lösung im Mittelpunkt des Interesses. Nach der Analyse dieser Schlüsselsituation ‚Projektbeginn‘ werden wir uns für den ‚Normalbetrieb‘ von offenen Programmierprojekten interessieren, indem wir der Frage nachgehen, wie über Zielsetzungen entschieden wird. Hier werden wir den grundlegenden Mechanismus der Entscheidungsfindung sowie einige Konsequenzen für die Gestaltung von Software herausarbeiten und die Rolle der Benutzer diskutieren. Es wird sich zeigen, daß – zumindest dem Fallbeispiel zu-

folge – diese Programmierprojekte trotz ihrer Offenheit über Abschließungsmechanismen verfügen, die den Einfluß von Anwendern auf die Definition von Zielsetzungen beschränken. Freie Softwareentwicklung bildet daher nicht generell ein Beispiel für gelungene partizipative Technikentwicklung. Im fünften, die Innenansichten abschließendem Kapitel werden wir die Frage stellen, wie es zu dem häufig mit freier Software in Zusammenhang gebrachten Merkmal Stabilität kommt. Unsere Antwort hierauf lautet: Die Stabilität ist Resultat der Rückkopplung zwischen Herstellungs- und Anwendungskontext und der vielfachen experimentellen Erprobung der Programme in heterogenen Anwendungskontexten. Die Gestalt von Software – hier die Stabilität – resultiert also aus der organisatorischen Struktur des Softwareentwicklungsprozesses.

Die Arbeit schließt mit einem vierten Teil, in dem wir den theoretischen Ertrag zusammenfassen. Hier werden wir zeigen, daß das Konzept der Innovationsnetzwerke zwar geeignet ist, einige wesentliche Merkmale von offenen Programmierprojekten zu beschreiben, allerdings an anderen zentralen Stellen unserer Untersuchung an ihre Grenzen stößt. Wir werden daher statt einer weitgehend implizit bleibenden Fundierung des ursprünglichen Konzepts auf einer ‚Rational-Choice Theorie‘ eine institutionalistische Lesart, basierend auf einem normativen Handlungsmodell, entwickeln.

Begriffliche Klärungen

Freie Software

Bereits die Benennung des hier interessierenden Typus von Software birgt großes Irritationspotential, werden doch von unterschiedlichen Seiten verschiedene Bezeichnungen vorgeschlagen und eine rege Diskussion um die ‚richtige' Bezeichnung des Gegenstands geführt. Wir beginnen daher in diesem Abschnitt mit einer Durchsicht der am weitesten verbreiteten Vorschläge ‚Free Software', ‚Open Source Software' und ‚Libre Software'[1], um dann zu begründen, weswegen wir im folgenden von ‚freier Software' sprechen werden.

Der Begriff ‚freie Software' ist der historisch älteste Begriff und geht auf das ‚Initial Announcement' des GNU-Projekts[2] vom 27.09.1983 zurück, das den Beginn der freien Softwareentwicklung markiert. Der Pionier Richard S. Stallman kündigte sein Projekt damals wie folgt an:

„Free Unix!
Starting this Thanksgiving I am going to write a complete Unix-compatible software system called GNU (for Gnu's Not Unix), and give it away free to

1 Daneben finden sich auch Kompromiß-Bezeichnungen, wie das etwas umständliche Free/Open Source Software (Narduzzo/Rossi 2003) oder das noch weniger elegante Free/Libre Open Source Software (Wichmann 2002; Ghosh/Robles/Glott 2002).

2 Das GNU-Projekt (Rekursives Akronym für GNU is not Unix) wurde mit der oben zitierten Ankündigung ins Leben gerufen. Der Gründer Stallman verfolgte und verfolgt mit dem Projekt weitreichende gesellschaftspolitischen Zielsetzungen, die uns weiter unten noch ausführlicher beschäftigen werden.

everyone who can use it. Contributions of time, money, programs and equipment are greatly needed.“[3]

In dieser Ankündigung wird die Absicht erklärt, ein Betriebssystem zu entwickelt, daß von jedermann benutzt werden kann. Das Charakteristikum ‚frei‘ steht hier also für das Recht zur Benutzung der Software, wobei allerdings offen bleibt, ob damit eher die Kostenlosigkeit der Benutzung oder die vom Autoren gegenüber dem Nutzer freizügig eingeräumten Rechte betont werden sollen. Diese Uneindeutigkeit der englischen Bezeichnung wird von Stallman später geklärt. In seinem prominenten Ausspruch „‚Free software‘ is a matter of liberty, not price. To understand the concept, you should think of ‚free‘ as in ‚free speech‘, not as in ‚free beer‘“,[4] macht der Autor deutlich, daß er Wert auf den Aspekt der Freizügigkeit der Verwendung legt und nicht auf die Kostenlosigkeit des Programms.

Eine alternative Bezeichnung für diesen Typus von Software tauchte mit ‚Open Source‘ erst im Jahre 1998 auf und machte der ursprünglichen Bezeichnung schnell Konkurrenz. ‚Open Source‘ verweist nun nicht auf Freiheiten, die Entwicklern und Nutzern an dem Programm eingeräumt werden, sondern bezieht sich auf die Zugänglichkeit des Quellcodes.[5] Begründet wird die Neuschöpfung des Begriffs von der Open Source Initative (OSI) wie folgt:

„But the real reason for the re-labeling is a marketing one. We’re trying to pitch our concept to the corporate world now. We have a winning product, but our positioning, in the past, has been awful. The term ‚free software‘ has been misunderstood by business persons, who mistake the desire to share with anticommercialism, or worse, theft.“[6]

3 Vgl. http://www.gnu.org/gnu/initial-announcement.html.

4 Vgl. The Free Software Definition (http://www.gnu.org/philosophy/free-sw.html). Wie wir weiter unten noch sehen werden, führt die Einräumung von weitgehenden Rechten an freier Software dazu, daß die Programme kostenlos sind. Bezahlt werden hingegen die unterschiedlichen mit freier Software verknüpften Dienstleistungen.

5 Im Fall von Software ist zwischen zwei Ausdrucksformen zu unterscheiden: Dem Quellcode des Programms, verfaßt in einer für Menschen les- und schreibbaren Programmiersprache, und dem Maschinencode, der auf Computern abgelaufen lassen werden kann. Während erstgenannter Code von der Syntax und Semantik her sich natürlichen Sprachen annähert, besteht der Maschinencode nur aus zwei Zeichen oder Signalen: ‚0‘ und ‚1‘ bzw. ‚Strom an‘ und ‚Strom aus‘.

6 Vgl. http://www.opensource.org/advocacy/case_for_hackers.php.

Mit der Neu- oder Umbenennung des Gegenstands wird also das Ziel verfolgt, die Mißverständlichkeiten der Bezeichnung ‚free software' zu umgehen und daraus resultierende Hindernisse auszuräumen, die einer weiteren Verbreitung dieser Form von Software entgegenstehen. Dabei werden aus der Besonderheit einer Zugänglichkeit des Quellcodes weitere Qualitäten der Software abgeleitet, die sie für unterschiedliche Gruppen attraktiv machen sollen.[7]

Neben diesen beiden Bezeichnungen wird vor allen Dingen im französischen Sprachraum ‚libre software' benutzt, die allerdings unter den Softwareentwicklern eher unüblich ist. Dalle/Jullien (2002: 3) favorisieren ‚libre Software', da die eingeräumten Freiheiten im Unterschied zu ‚Open Source' betont und die Doppeldeutigkeit des englischen Adjektivs ‚free' gleichzeitig vermieden wird.

Gegen sämtliche Bezeichnungen lassen sich nun Einwände vorbringen, wie die weitere Diskussion gezeigt hat: Der Terminus ‚free software' besitzt die bereits erwähnte Doppeldeutigkeit, wobei das Verständnis von ‚frei' im Sinne von ‚kostenlos' nicht geeignet ist, den hier interessierenden Typus von Software abzugrenzen. So ist beispielsweise der Microsoft Internet Explorer kostenlos erhältlich, bildet aber gerade kein Beispiel für den hier gemeinten Typ.[8] Weiter ist gegen den Begriff einzuwenden, daß er aufgrund seiner Geschichte stark mit normativen Implikationen beladen ist: Er basiert auf der Unterscheidung ‚frei' / ‚unfrei' und wird insbesondere von der Free Software Foundation (FSF) genutzt, um auf die Abhängigkeitsbeziehungen hinzuweisen, die nach ihrer Perspektive durch die Anwendung von proprietären Programmen[9] zum Hersteller entstehen, hinzuweisen und freie Softwareentwicklung als emanzipatorische Bewegung zu interpretieren.[10]

Demgegenüber besteht der Nachteil der Bezeichnung ‚Open Source Software' in seiner Fokussierung auf die Zugänglichkeit des Quellcodes: So wird er mißbräuchlich zur Bezeichnung von Programmen benutzt, deren Quellcode zwar zugänglich ist, die aber von den Lizenzbedingungen her weder den in der Open Source-Definition, noch den in der Free Software-Definition genannten Distinktionsmerkmalen entsprechen. Die

7 Vgl. http://www.opensource.org/advocacy/case_for_hackers.php, http://www.opensource.org/advocacy/case_for_customers.php, http://www.opensource.org/advocacy/case_for_business.php.

8 Hierauf weist die Open Source Initiative zu Recht hin. Vgl. http://www.opensource.org/advocacy/free-notfree.php.

9 Der Begriff wird hier provisorisch eingeführt und im anschließenden Abschnitt erläutert.

10 Vgl. hier insbesondere den Aufsatz von Stallman „Why Software Should Be Free" (1992). Als Online-Dokument: http://www.fsf.org/philosophy/should befree.html.

Bezeichnung bildet ein Einfallstor für Unternehmen, welche die von der Marketing-Initiative OSI in Zusammenhang mit ‚Open Source' gebrachten Merkmale von Software nutzen, um ihre Produkte zu bewerben, ohne das die von ihnen vertriebene Software die notwendigen Kriterien tatsächlich erfüllt.

Der entscheidende Nachteil von ‚Libre Software' bildet, wie erwähnt, seine geringe Verbreitung und Bekanntheit. Daneben ist er gegenüber den Begriffen ‚Free Software' und ‚Open Source' weit weniger eindeutig definiert. Da sich durch keinen der genannten Bezeichnungen Ambiguitäten vermeiden lassen, werden wir im folgenden die deutsche Übersetzung der historisch ältesten und am weitesten verbreiteten Bezeichnung ‚freie Software' nutzen, allerdings ohne die von der FSF betonte normative Interpretation gleichzeitig übernehmen zu wollen. Die deutsche Übersetzung hat den Vorteil, daß das Charakteristikum ‚frei' weniger als das amerikanische ‚free' Kostenlosigkeit meint und die Möglichkeit des Mißverstehens hier geringer ist als beim amerikanischen Pendant. ‚Freie Software' bildet für uns lediglich eine Bezeichnung für einen besonderen Typus von Software, mit dessen Charakteristika wir uns nun beschäftigen wollen.

Die Besonderheiten des Typus ‚freie Software' erschließen sich durch die Softwarelizenzbedingungen, die den Umgang mit Software regulieren und die nicht nur für die Distribution und Nutzung weitreichende Folgen haben, sondern – wie wir im Verlaufe der Arbeit noch sehen werden – gerade auch für die Entwicklung der entsprechenden Programme. Um die Funktionsweise dieser vertraglichen Regulierung zu verstehen, ist es notwendig, Software wenigstens kurz aus rechtlicher Perspektive zu betrachten.

Software wird vom Gesetzgeber als Gut betrachtet, das von überragender wirtschaftlicher Bedeutung ist und als schutzwürdig gilt.[11] Ungewöhnlich an der rechtlichen Regulierung von Software ist, daß sie unter zwei Gesetze, einerseits das Urheberrecht, andererseits das Patentrecht, fällt.[12] Der Anwendung von beiden Rechte auf Software weist auf

11 Die wirtschaftliche Bedeutung und die davon abgeleitete besondere Schutzwürdigkeit werden in nahezu jeder Publikation zum Marken-, Patent- und Urheberrecht von Software betont. Vgl. z.B. den Richtlinienvorschlag der Kommission der Europäischen Gemeinschaft über die Patentierbarkeit computerimplementierter Erfindungen (http://europa.eu.int/comm/internal_ market /en/indprop/com02-92de.pdf).

12 Abgesehen von der Ausnahme ‚Software' verhalten sich die beiden Bereiche des Immaterialgüterrechts wechselseitig exklusiv: Güter, die unter das Urheberrecht fallen, wie z.B. Romane, Theaterstücke oder Gemälde, werden nicht durch das Patentrecht, Güter, die unter das Patentrecht fallen,

ein Einordnungsproblem in die Rechtsdogmatik hin: Durch das Urheberrecht werden üblicherweise Werke von Autoren geschützt, denen der Gesetzgeber einen literarischen, künstlerischen oder wissenschaftlichen Wert beimißt. Aus der Perspektive des Urheberrechts wird Software als Schriftwerk verstanden, also als eine Aneinanderreihung von Zeichen gemäß der Semantik, Syntax und Grammatik einer formalen Programmiersprache.[13]

Der gesetzlich gewährte Patentschutz bezieht sich hingegen nicht auf die Realisierungsform, sondern auf die einem Programm zugrunde liegenden technischen Ideen (Endres 2001: 19).[14] Während die Patentierung von Software in Japan und den USA möglich ist, gilt dies für die Länder der EU bislang nicht. In europäischen Ländern sollen nach der aktuellen Rechtslage Patente nur dann erteilt werden, sofern sich die Erfindung durch Originalität, eine gewisse Erfindungshöhe, wirtschaftliche Relevanz und einen ‚technischen' Bezug auszeichnet. Das letztgenannte Kriterium ist dann erfüllt, sobald ein technisches Problem mit Hilfe von technischen Mitteln gelöst wird. In der Anwendung des Patentrechts führt dies nun allerdings häufig zu Schwierigkeiten, da es zwar in den meisten Fällen unstrittig ist, ob das vorliegende Programm eine Lösung offeriert, während das technische Problem sich meist durch eine geringere Eindeutigkeit auszeichnet und schlecht zu bestimmen ist (vgl. ebd. 20). Diese Unsicherheit soll auf europäischer Ebene beseitigt werden, wobei die EU-Kommission einen Richtlinienvorschlag ausgearbeitet hat, der einen Patentschutz für Erfindungen vorsieht, die sich auf einen Computer oder vergleichbare Vorrichtungen stützen, durch Ablauf eines

wie z.B. technische Verfahrens- oder Produkterfindungen, nicht durch das Urheberrecht geschützt.

13 Vgl. Lutterbeck/Gehring/Horns 2001. In dieser Einordnung wird Software als Text verstanden und damit eine linguistische Perspektive eingenommen. Hierzu Rehbinder (1998: 81): „Geschützt sind also alle Werke, die sich der Sprache als Ausdrucksmittel bedienen, gleichviel ob sie schriftlich niedergelegt sind oder nicht. Ein Sprachwerk ist nur dann gegeben, wenn ein ‚begrifflicher Inhalt' (Gedanke) durch die Sprache ausgedrückt wird. […] Gleichgültig ist auch, welche Sprache gewählt wurde, ob die inländische oder eine fremde, ob eine lebende oder eine tote, ob eine Kunstsprache wie Esperanto oder eine Computersprache wie Algol usw., selbst die Mitteilung des Inhaltes in Zahlen, Formeln oder in einer Bildersprache genügt."

14 Zur Abgrenzung von Urheber- und Patentrecht siehe auch Haberstumpf 1996: 5 f. Der patentrechtliche Schutz bezieht sich auf Erfindungen, mit denen naturwissenschaftliche Erkenntnisse in technische Handlungsweisen umgemünzt werden. Schutzgegenstand stellt hier also ein Mittel dar, daß zu bestimmten Zwecken eingesetzt werden kann. Während der urheberrechtliche Schutz kraft Gesetz jedem geschaffenen Werk zukommt, setzt der patentrechtliche Schutz einen staatlichen Verleihungsakt voraus.

Programms realisiert werden und die als technischen Beitrag zu werten sind, der für eine fachkundige Person nicht naheliegend ist.[15] Im folgenden werden wir auf den patentrechtlichen Schutz von Software nicht weiter eingehen, sondern uns ausschließlich auf die Darstellung des Urheberrechts und der darauf basierenden Softwarelizenzen beschränken, um darauf aufbauend eine Typologie zu entwickeln.

Während es lange unklar war, ob Software einen urheberrechtlichen Schutz zukommt, werden seit der Urheberrechtsnovelle vom 24.06.1985 in Deutschland Programme für die Datenverarbeitung als Sprachwerke verstanden, die in den Gegenstandsbereich des Urheberrechts fallen.[16] Durch diese Novellierung wurden allerdings nur Programme geschützt, die das Können eines Durchschnittsprogrammierers überstiegen (Moritz 1986: 27). Eine Ausweitung des Schutzes von Computerprogrammen, fand durch die Novelle vom 24.06.1993 statt. Seitdem wird das Urheberrecht auf Programme angewendet, sofern sie Ergebnis einer eigenen geistigen Schöpfung des Urhebers darstellen. Eine überragende Gestaltungshöhe wird also nicht mehr verlangt (Raubenheimer 1994: 71).[17] Hiermit setzte der deutsche Gesetzgeber die Richtlinie zur Harmonisierung des Urheberrechts in der Europäischen Gemeinschaft um, in der die Ausdehnung des urheberrechtlichen Schutzes auf Programme für die Datenverarbeitung gefordert wurde.

Das europäisch harmonisierte Urheberrecht gewährt dem Werkschöpfer und seinen Nachkommen einen 70 Jahre über den Tod des Autors hinausgehenden Schutz des Werks (vgl. Rehbinder 1998: 55 ff.). Dieser erstreckt sich auf die Rechte zur Verwertung des Werks, wobei damit Vervielfältigung-, Verbreitungs-, Wiedergabe- und Senderechte eingeschlossen werden. Ebenfalls geschützt wird die Integrität des Werks, so daß Veränderungen nur mit Einwilligung des Werkschöpfers bzw. seiner Nachkommen vorgenommen werden dürfen.

International werden Werke durch multilaterale Vereinbarungen geschützt. Hierzu zählt an erster Stelle die ‚Revidierte Berner Übereinkunft' (RBÜ), die eine Inländerbehandlung vorsieht. Damit ist die Anwendung des jeweils in einem Land geltenden Urheberrechts auf im Ausland erstellte Werke gemeint.[18] Eine weitere wichtige internationale

15 Vgl. den Richtlinienvorschlag der Kommission der Europäischen Gemeinschaft über die Patentierbarkeit computerimplementierter Erfindungen (http://europa.eu.int/comm/internal_market/en/indprop/com02-92de.pdf).

16 Vgl. Moritz 1986.

17 Pres (1994a: 520) zufolge hat die Novellierung die Konsequenz, daß sämtliche Programme, die nicht nur Vorbekanntes oder Fremderstelltes enthalten und nicht trivial sind, urheberrechtlichen Schutz genießen.

18 Computerprogramme wurden bereits mit der Urheberrechtsnovelle vom 24.06.1985 unter urheberrechtlichen Schutz gestellt (Moritz 1986: 19).

Vereinbarung bildet das am 01.01.1995 in Kraft getretene ‚Agreement on Trade-Related Aspects of Intellectual Property Right' (TRIPS). Die Übereinkunft schreibt die Behandlung von Software und Datenbanken als Werk der Literatur vor und fordert damit von den Unterzeichnerstaaten die Anwendung des jeweiligen Urheberrechts auf Computerprogramme. TRIPS geht insofern über die RBÜ hinaus, als daß zum einen der Kreis der Unterzeichnerländer größer ist, zum anderen Datenbanken und Computerprogramme explizit in der Definition des Anwendungsbereichs genannt werden und damit mögliche Unklarheiten bei nationalen Urheberrechtgesetzen im Fall von Software beseitigt sind (vgl. Lehmann 1996: 3). Darüber hinaus stellt auch das am 20.12.1996 verabschiedete ‚WIPO Copyright Treaty'[19] klar, daß Computerprogramme in sämtlichen Formen in den Geltungsbereich der RBÜ fallen und das Prinzip der Inländerbehandlung angewendet werden muß (v. Lewinsky 1997: 442).

Die Verwertung der oben genannten Rechte findet durch Lizenzen statt, mit denen ein Lizenznehmer von einem Lizenzgeber bestimmte Nutzungsrechte erwirbt. Hierbei ist zu unterscheiden zwischen der Übertragung von einfachen und ausschließlichen Nutzungsrechten, wobei das einfache Nutzungsrecht bestimmte Nutzungsrechte an einen Lizenznehmer überträgt, welches Dritte an der Werknutzung nicht ausschließt, während im Fall des ausschließlichen Nutzungsrechts eine weitere Übertragung von Nutzungsrechten an andere nicht möglich ist (vgl. Fischer 1993: 5 f.). Eine ausschließliche Lizenz ist in Fällen üblich, in denen ein Werkschöpfer einer Verwertungsgesellschaft seine Rechte überträgt. Gebrauchslizenzen können unterschiedliche Formen annehmen: So ist es beispielsweise möglich, nur bestimmte Formen der Verwendung zuzulassen, die Verwendung der Software an weitere Bedingungen zu knüpfen oder diese zeitlich zu befristen. Entscheidend für den hier interessierenden Gegenstand ‚freie Software' sind nun vier Dimensionen, die durch Softwarelizenzen geregelt werden:

1. Die durch die Lizenz eingeräumten Rechte zum *Ablaufenlassen des Programms*: Auf den ersten Blick scheint es offensichtlich zu sein, daß Softwarelizenzen das Recht zur Benutzung des Programms, also der Erstellung einer Kopie im Arbeitsspeicher des Computers, sowie der Anwendung der Algorithmen auf Daten beinhalten müssen, damit der Anwender einen bestimmten Nutzen aus dem Erwerb einer Lizenz zieht. Dies muß allerdings nicht der Fall sein. So kann ein Händler beispielsweise eine Vertriebslizenz erwerben, die Vervielfältigung und Unterlizensierung erlaubt, nicht aber die Anwendung des Programms (vgl. Pres

Die Urheberrechtschutzfähigkeit beschränkte sich allerdings auf Programme, die das Durchschnittskönnen ‚deutlich überragen' (ebd. 27).

19 WIPO = World Intellectual Property Organization.

1994b: 523). Das Recht zum Ablaufenlassen des Programms kann dabei an zeitliche, räumliche oder anwendungsmäßige Restriktionen gebunden sein, wobei letzteres beispielsweise eine bestimmte Datenverarbeitungsanlage oder ein bestimmtes Einsatzgebiet meinen kann.[20]

2. Lizenzbestimmungen bezüglich der *Vervielfältigung von Software*. Auch die Vervielfältigung von Software wird durch Lizenzen vertraglich geregelt, wobei hier ebenfalls unterschiedliche Gestaltungsmöglichkeiten existieren. So erlauben typische Endanwenderlizenzen lediglich die Erstellung einer temporären Kopie des Programms im Arbeitsspeicher und die Anfertigung einer Sicherheitskopie, während bei sogenannter Shareware die Vervielfältigung von Software oftmals nicht beschränkt wird, so daß beliebig viele Programmkopien erzeugt werden können.

3. *Einseh- und Veränderbarkeit des Quellcodes*: Eine Vielzahl von Programmen wird ausschließlich im binären Maschinencode ausgeliefert, so daß der Quellcode nicht zugänglich ist. Eine Möglichkeit, den Maschinencode in den Quellcode zu überführen, ist das sogenannte reverse engineering bzw. die Decompilierung, wobei generelle Decompilierungsverbote in den Lizenzbedingungen des betreffenden Programms nach der EU-Richtlinie unwirksam sind. Allerdings beschränkt der Gesetzgeber das Recht zur Decompilierung nur auf den Fall der Herstellung von Interoperabilität zwischen dem lizensierten und einem anderen zu entwickelnden Programm (Moritz 1993a: 265). Weiter erlaubt die EU-Richtlinie auch Lizenzklauseln, die eine Bearbeitung, Übersetzung und Umgestaltung des Programmquellcodes untersagen. Daneben finden sich auch Programme, die sowohl im Quell- als auch im Maschinencode ausgeliefert werden und deren Lizenzen sowohl die Einsicht des Quellcodes als auch dessen Veränderung zulassen. Die Gewährung dieser Rechte geht also über die von den europäischen Gesetzgebern gestellten Mindestanforderungen weit hinaus.

4. *Verbreitung des Programms*: Die EU-Richtlinie erlaubt Lizenzklauseln, die dem Lizenznehmer eine Unterlizensierung des Programms untersagen (Moritz 1993a: 265; 1993b: 415). Auch hier finden sich Softwarelizenzen, die dem Empfänger wesentlich großzügigere Rechte ein-

20 Eine typische Einschränkung bildet eine Klausel, welche die Nutzung des Programms nur auf einer bestimmten Datenverarbeitungsanlage erlaubt, die üblicherweise durch Fabrikat-, Modell- und Seriennummer gekennzeichnet ist (Moritz 1993a: 263). Ein Beispiel hierfür bilden die OEM (Original Equipment Manufacturer)-Lizenzen, die aber nach europäischem Recht bei Behinderung anderer Hardwarehersteller als Wettbewerbsverstoß auf dem europäischen Markt angesehen werden (Moritz 1993b: 346).

räumen. Ein Beispiel hierfür bildet wiederum Shareware, die von jedermann kopiert und meist auch in Verkehr gebracht werden darf.

Auf der Basis der vier unterschiedlichen durch Softwarelizenzen eingeräumten Rechte bezüglich der Anwendung, Vervielfältigung, Verbreitung und Modifikation können wir nun die auf dem Markt vorfindbare Software typologisieren:

Softwaretyp	**Anwendung**	**Vervielfältigung**	**Verbreitung**	**Modifikation**
proprietär (zum Normalgebrauch)	+	-[21]	-	-
proprietär (Händlerlizenz)	(-)	+	+	-
Shareware	+	+	(+)	-
Public Domain	+	+	+	+
Freie Software	+	+	+	+

Tabelle 1: Typologie der Softwarelizenzen

Wir unterscheiden in der tabellarischen Zusammenstellung fünf Typen von Software, wobei im Fall von proprietärer Software übertragene Rechte aufgelistet werden, wie man sie typischerweise in Händler- und Endnutzerlizenzverträgen findet. Selbstverständlich finden sich hier auch Differenzen, wie beispielsweise zeitliche und räumliche Nutzungsrestriktionen (Pres 1994b: 523), aber auch – gerade im Bereich der konfektionierten Software (Auftragsarbeiten) – Lizenzbedingungen, welche die Einsicht und Modifikation des Quellcodes zur Weiterentwicklung oder Anpassung des Programms zulassen.

Nach dieser systematischen Arbeit sind wir nun in der Lage, freie Software von anderen Typen abzugrenzen. ‚Freie Software' sind all diejenigen Programme, deren Lizenz die Nutzung, Vervielfältigung, Verbreitung *und* Veränderung uneingeschränkt zulassen. Damit bildet auch der in der Tabelle separat genannte Typus Public Domain-Software eine Unterkategorie von freier Software. Das bekannteste, weit verbreitetste[22] und älteste Beispiel für freie Softwarelizenzen bildet die GNU General Public License (GNU GPL), in der die vier Rechte durch die folgenden Klauseln gewährt werden:

21 Erlaubt ist das Laden und Ablaufenlassen des Programms, also die Erstellung einer Programmkopie im Arbeitsspeicher des Computers sowie die Erstellung einer Sicherheitskopie (vgl. Pres 1994b: 523).

22 Lerner/Tirole (2002a) kommen zu dem Schluß, daß in 72 Prozent der Fälle ihres Samples die GPL Anwendung findet. Ähnliche Zahlen finden sich auch in der Untersuchung von Bonaccorsi/Rossi (2003: 1249).

„You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program. […] You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1."[23]

Die Besonderheit dieser Softwarelizenz liegt nicht nur in der großzügigen Gewährung von Nutzungsrechten, sondern in der einzigen Einschränkung, welche die Lizenz vornimmt: Mit der sogenannten Repetitionsklausel wird die Veränderung der Lizenzbedingungen für sämtliche modifizierte und nicht-modifizierte Kopien des Programms untersagt, ein Charakteristikum, das ‚copyleft'-Software von Public Domain-Software unterscheidet.[24] Im Fall des letztgenannten Typus ist es möglich, die Lizenzbedingungen hinzuzufügen und damit die Nutzungsrechte von künftigen Programmempfängern zu beschneiden. Demgegenüber sorgt das in der GNU GPL ausgesprochene Verbot der Veränderung von Lizenzbedingungen dafür, daß sämtliche Empfänger des Programms (und dessen Weiterentwicklungen) dieselben großzügigen Nutzungsrechte erhalten wie der Empfänger des ursprünglichen Programms. Neben der GNU GPL existieren im Bereich freier Software weitere freie Softwarelizenzen; Beispiele sind die Lesser General Public License (LGPL), die Berkeley Software Distribution License (BSD License), die modified BSD License, die Apache License, die Mozilla Public License (MPL), die Netscape Public License (NPL) und die Qt Public License (QPL).[25]

23 GNU General Public License (GNU GPL) (http://www.fsf.org/licenses/gpl.html).

24 Bei Public Domain-Software handelt es sich um nicht urheberrechtlich geschützte Software bzw. solche, bei dem der Autor auf das Copyright verzichtet. Dies ist jedoch nur nach amerikanischem, nicht aber nach europäischem Urheberrecht möglich.

25 Wir beschränken uns auf die Nennung der bekanntesten freien Softwarelizenzen. Für eine weitere Übersicht siehe: http://www.gnu.org/philosophy/license-list.html.

Freie Softwareprojekte

Ebenso ungewöhnlich wie die im letzten Abschnitt dargestellten freien Softwarelizenzen mutet auch die soziale Struktur an, in der freie Software typischerweise entwickelt wird und die sich auf den ersten Blick den in der Soziologie üblichen Kategorien wie Organisation, Gruppe und Interaktionssystem sperrt. Wir beginnen daher mit einer Typologisierung der am Entwicklungsprozeß beteiligten Akteure von ‚Core-Entwicklern', ‚sporadischen Entwicklern', ‚Pilotanwendern', ‚Normalnutzern' sowie verschiedenen Arten von Organisationen.

Am auffälligsten innerhalb des Entwicklungsnetzwerks ist der Kreis von Personen, der die Entwicklung selbst vorantreibt, also eigene Beiträge zum Programm leistet. Die Aktivität setzt die Kenntnis einer oder mehrere Programmiersprachen voraus. Es handelt sich um eine Gruppe von Personen, die im Bereich der Software- und Hardwaretechnik versiert ist und die zumindest zum Teil aufgrund ihrer profunden und wenig alltäglichen Kompetenzen als technische Elite bezeichnet werden kann.

Diese Gruppe der Entwickler bildet nun allerdings keine homogene Einheit, sondern unterscheidet sich maßgeblich hinsichtlich ihrer Kompetenzen, ihrer beruflichen Tätigkeit und des Umfangs ihres Engagements. Auf der Ebene der Kompetenz sind maßgebliche Differenzen hinsichtlich der Programmiererfahrungen, der Kenntnis des Quellcodes der zu entwickelnden Software sowie der beigesteuerten Beiträge zu beobachten.

Unter den Entwicklern von freier Software finden sich sowohl Personen, welche die Entwicklung hauptberuflich vorantreiben, als auch solche, deren Beruf nicht mit der Entwicklung und Anwendung von freier Software in Verbindung steht. Empirische Befunde aus anderen Untersuchungen zeigen dabei, daß die letztgenannte Gruppe die zahlenmäßig stärkere Fraktion bildet. Der größte Teil des Codes stammt allerdings von einer kleinen Zahl hoch aktiver Entwickler (Ghosh/Prakash 2000; Ghosh/Robles/Glott 2002: 14; Koch/Schneider 2002: 31).

Unterschiede treten auch in bezug auf das Engagement, also der Zeit und Kosten, auf, die von den Programmierern für die Entwicklung aufgewendet wird. Das Spektrum reicht hier von einer jahrelang währenden Beteiligung an einem Projekt, bei der mit großer Kontinuität eigene Beiträge zur Entwicklung beigesteuert werden, bis zu einem sporadischen oder einmaligen Beitrag zu einem Programmierprojekt.

Die Gruppe der Entwickler ist also in einem weit geringeren Maße homogen, als es die Kategorisierung erwarten ließe. Aufgrund dieser Differenzen wollen wir zwischen den Entwicklern, die kontinuierlich ein Projekt vorantreiben, und sporadisch beteiligten Entwicklern unterschei-

den.[26] Die erste Gruppe der *Core-Entwickler* ist über einen längeren Zeitraum[27] kontinuierlich an einem Projekt beteiligt und steuert maßgebliche Beiträge zur Entwicklung des Projekts bei. Sie zeichnen sich weiter durch eine intime Kenntnis des Quellcodes aus und arbeiten meist an zentralen Bestandteilen des Programms.

Die Gruppe der *sporadischen Entwickler* zeichnet sich demgegenüber durch Diskontinuität aus – unter Umständen kann sich ihr Engagement auf einen einmaligen Beitrag beschränken. Häufig resultiert der Beitrag dann aus eigenen Anwendungsbedürfnissen, also bei der Nutzung auftauchenden Probleme oder fehlenden Funktionen, die sporadische Entwickler dann beseitigen beziehungsweise ergänzen. Diese Personengruppe fluktuiert häufig zwischen verschiedenen Projekten, engagiert sich kurzfristig in einem Projekt, um wenig später einen Beitrag zu einem anderen Projekt beizusteuern.

Während in der Frühphase von freier Software bis ca. 1993 die Gruppe von Entwicklern und von Anwendern nahezu identisch waren, es also lediglich eine geringe Anzahl an Personen gab, welche die Software ausschließlich anwendete und nicht zur Entwicklung beitrug, ist durch die Verbreitung von freier Software ab Mitte der 1990er Jahre eine wachsende Gruppe von Personen entstanden, welche die Software ausschließlich anwendet. In bezug auf die Entwicklung bilden Nutzer ebenfalls keine homogene Gruppe, sondern zerfallen in *Pilotanwender* und *Normalnutzer*. Pilotanwender verfügen über ein hohes Maß an technischer Kompetenz, installieren die jeweils aktuellsten, gegebenenfalls noch in der Entwicklung befindlichen Versionen eines Programms und erproben diese. Ihr Beitrag zur Entwicklung liegt vorrangig in der Meldung von Programmfehlern (Bugs), die im Regelfall noch vor der Veröffentlichung einer neuen Version von den Entwicklern beseitigt werden. Normalnutzer bilden hier eine Residualkategorie, die sämtliche Anwender von freier Software umfaßt, welche ausschließlich die als stabil deklarierten Versionen des Programms nutzen. Da sich nur ein geringer Bruchteil dieser Personen an der Entwicklung beteiligt, ist ihre Größe schwer einzuschätzen. Trotzdem finden sich auch aus der Gruppe der Normalnutzer Personen, die sich an der Entwicklung beteiligen, typischerweise entweder in Form der Nachfrage nach einer bestimmten Funktionalität (sogenannten feature-requests) oder ebenfalls in Form der Meldung von Fehlern und Problemen des Programms.

Eine weitere für die Entwicklung von freier Software wichtige Akteursgruppe bilden die Distributoren. Hierbei handelt es sich meist um

26 Vgl. zur folgenden Unterscheidung auch Grassmuck 2002: 237.

27 Hierunter verstehen wir eine mehrere Monate währende Beteiligung.

Unternehmen[28] – also Organisationen – die eine Sammlung von freier Software zusammenstellen und vertreiben.[29] Im Vergleich zu Softwareunternehmen, die proprietäre, selbstentwickelte Software verkaufen, ergeben sich aus den unterschiedlichen Lizenzbedingungen, denen die Software jeweils unterliegt, verschiedene Geschäftsmodelle: Während das Geschäftsmodell von Unternehmen, die proprietäre Software vertreiben, auf dem Verkauf von Urheberrechten basiert (gegen die Zahlung einer Lizenzgebühr werden dem Lizenznehmer bestimmte Formen der Nutzung eingeräumt), lassen freie Softwarelizenzen dies gerade nicht zu. Im Unterschied zu klassischen Softwareunternehmen oder Softwarehäusern verfügen Distributoren nicht über exklusive Verfügungsrechte, die durch einen Vertrag übertragen werden könnten. Das Geschäftsmodell von Distributoren gründet sich daher auf mit der Software verknüpfte Dienstleistungen, wie der Zusammenstellung und Verbringung der Software auf CD-ROM, Dokumentationen, Installations- sowie Anwendungssupport und anderes.

Die Tätigkeit der Distributoren beschränkt sich jedoch nicht – wie die bisherige Beschreibung nahelegen könnte – auf den Vertrieb von freier Software. Sie beteiligen sich auch an der Entwicklung, indem sie entweder Programmierer mittels eines Sponsorenmodells monetär unterstützen oder eigene Entwicklungsabteilungen unterhalten. Darüber hinaus stellen sie Speicherkapazität auf Webservern für Softwareentwicklungsprojekte zur Verfügung und unterstützen diese auch durch Hardwarespenden oder der Bereitstellung von Rechenzeit auf schwer zu beschaffender oder teurer Hardware. Die besondere Leistung von Distributoren für die Entwicklung von freier Software liegt also vorrangig in der Mobilisierung von monetären, personellen und anderen materiellen Ressourcen, die sich allerdings auf eine recht übersichtliche Zahl prominenter und erfolgreicher Projekte beschränkt.

Neben den genannten drei Gruppen existieren weitere Akteure oder Akteursgruppen, die schwierig zu klassifizieren und einzuordnen sind,

---

28 Eine Ausnahme bildet hier das Projekt Debian, ein Zusammenschluß von freiwilligen Personen, deren Ziel im Vertrieb von freien Softwaresystemen liegt (vgl. http://www.debian.org).

29 Die bekanntesten sind hier: Red Hat (http://www.redhat.com), SuSE (http://www.suse.de), Slackware (http://www.slackware.com), Mandrake (http://www.linux-mandrake.com), VA Linux (http://www.valinux.com), CorelLinux (http://www.linux.corel.com), Caldera (http://www2.caldera-systems.com), Linux PPC (http://www.linuxppc.org), Conectiva (http://en.conectiva.com), Turbo Linux (http://www.pht.com) und Stampede (http://www.stampede.org). Eine kommentierte Übersicht über Linux-Distributionen bietet die Webseite von Buchholz: http://www. kefk.net/linux/Distributionen/Typologie/index.asp.

da es sich hier um Einzelpersonen und Organisationen handelt. Diese Akteure treten im Regelfall auf Webseiten in Erscheinung und es läßt sich nur vermittels umfangreicherer Recherchen feststellen, um was für eine Art von Akteur es sich im einzelnen handelt. Wir wollen im folgenden kurz einige zentrale Organisationen vorstellen und diese Akteursgruppe als ‚entwicklungsnahe Organisationen' kategorisieren:

Programmiererorganisationen, die über Softwarelizenzen und derer Bedeutung informieren und/oder diese ausarbeiten: Diese Organisationen beschäftigen sich mit den rechtlichen Grundlagen der Entwicklung, Verbreitung und Anwendung von freier Software, achten auf die Einhaltung der Lizenzbedingungen, informieren Entwickler über die für sie günstigen Lizenzen und verfolgen meist auch das Ziel, eine größere Öffentlichkeit über freie Software zu informieren und für deren Anwendung zu werben. Beispiele sind hier die Free Software Foundation (FSF), die Open Source Initiative (OSI), Debian und Opencode.[30]

Einen wichtigen Stellenwert für die Berichterstattung und Verbreitung von Informationen über technische und nichttechnische Aspekte der Entwicklung und Anwendung von freier Software nehmen *Printmagazine* und *Websites* ein, die sich in ihrer Berichterstattung thematisch auf freie Software beschränken. An erster Stelle zu nennen sind hier Linux Gazette, Linux Journal, Linux Today, Linux Weekly News, Slashdot und Linux Magazin.[31]

Eine wichtige Orientierung unter den sich auf freie Software beziehenden Angeboten, Gruppen und Projekten innerhalb des Internet bieten die sogenannten *Portale*. Der Schwerpunkt dieser z.T. sehr umfangreichen Seiten liegt in geringerem Maße auf der Berichterstattung über aktuelle Entwicklungen, sondern auf der Vernetzung und Verlinkung der online verfügbaren Angebote. Hier finden sich Verweise auf die unterschiedlichen Entwicklungsprojekte, Downloadmöglichkeiten, Hinweise auf Softwaredokumentationen und Diskussionsforen. Beispiele für solche Portale bilden Linux Links, Linux Search und Linux Search Engine.[32]

Zusammengenommen bilden diese unterschiedlichen Akteure eine abwechslungsreiche Landschaft, die sich einer einfachen Einordnung

30 http://www.fsf.org, http://www.opensource.org, http://www.my-open source.org, http://www.debian.org/social_contract.de, http://eon.law.har vard.edu/opencode, Stand 04/2004.

31 http://www.linuxgazette.com, http://www2.linuxjournal.com, http://www. linuxtoday.com/index.html, http://www.lwn.net, http://www.slashdot.org, http://www.linux-magazin.de.

32 http://www.linuxlinks.com, http://www.linuxsearch.de, http://www.fokus. gmd.de/linux/linuxsengines.html.

widersetzt: Es handelt sich hierbei nicht um eine Organisation, da das Kriterium von formalen Mitgliedschaftsregeln ebenso fehlt wie eine Zugangskontrolle zu dieser Struktur. Von einem Interaktionssystem unterscheidet es sich nicht nur durch die hohe Stabilität in der Zeitdimension, sondern auch dadurch, daß nicht sämtliche Elemente miteinander interagieren. Die Einordnung der Struktur unter die Kategorie der Gruppe fällt aus, da sie nicht nur Personen, sondern auch Organisationen umfaßt. Ähnliche Probleme ergeben sich bei der Bezeichnung ‚Community‘, die von manchen Autoren allerdings gern genutzt wird (Scacchi 2003; Tzouris 2002; Madanmohan/Navelkar 2002; Ciborra/Andreu 2001).[33]

Weiter trägt der Begriff ‚Netzwerk‘, da er flexibel genug ist, um eine soziale Struktur, die sich von ihren Elementen her als stark heterogen darstellt, zu erfassen. Wir wollen die soziale Struktur, in der freie Software entwickelt wird, daher als ‚Netzwerk freier Softwareentwicklung‘ bezeichnen, dabei allerdings betonen, daß es sich weder um ein personales[34] noch um ein interorganisationales Netzwerk[35] handelt. Vielmehr zeichnet es sich gerade dadurch aus, daß es sich aus unterschiedlichen Elementen zusammensetzt und gleichzeitig eine große Heterogenität hinsichtlich der Beziehungen unter den Elementen aufweist.
Das Netzwerk freier Softwareentwicklung zerfällt in eine Vielzahl kleinere Netzwerke, in denen die Entwicklungstätigkeit stattfindet: freie Softwareentwicklungsprojekte, die ihrerseits personale Netzwerke darstellen. Elemente bilden hier Personen und nicht Organisationen, wobei ausschließlich die weiter oben unterschiedenen Akteure Core-Entwickler und sporadische Entwickler, Pilot- und Normalanwender sowie ein sogenannter Maintainer, der üblicherweise selbst ein Core-Entwickler und offizieller Leiter des Projekts ist, anzutreffen sind. Freie Softwareentwicklungsprojekte zeichnen sich durch zwei Merkmale aus: Zum einen findet die Entwicklung nicht an einem räumlich lokalisierbaren Ort statt, zu dem man sich begeben und an dem man die Entwicklungstätigkeit beobachten könnte. Die Beteiligten sind über den Globus verteilt und koordinieren sich vermittels unterschiedlicher Medien wie dem WWW, Mailinglisten und dem Internet Relay Chat (IRC). Von außen betrachtet treten uns freie Softwareentwicklungsprojekte als Kommunikationsplatt-

33 Hierbei handelt es sich um einen Begriff, der von Howard Rheingold (1992, 1994) in die Diskussion gebracht wurde und sich in der Literatur zu computervermittelten Kommunikation großer Beliebtheit erfreut. Vgl. Baym 1995b und Jones 1995, 1996. Zu einer ausführlichen Kritik dieses problematischen Begriffs siehe Mitra 1999 und Paccagnella 1997.

34 Im Sinne von Welllmann 2000; Garton/Haythornthwaite/Wellmann 1997.

35 Vgl. zu dieser Form von Netzwerken Powell 1996 und Mayntz 1992.

formen entgegen, die sich aus den folgenden Bestandteilen zusammensetzen:

Webseiten: Sie dienen der Außendarstellung des Projektes. Dort werden üblicherweise Informationen über den Stand der Entwicklung des Programms, Bezugsquellen von aktuellen und älteren Versionen, Möglichkeiten zur Beteiligung an der Entwicklung und die für das Projekt wichtigsten Personen mitgeteilt.

Eine oder (je nach Größe des Projekts) mehrere Mailinglisten. Auf den Mailinglisten findet die Kommunikation zwischen den Entwicklern sowie an der Entwicklung und Anwendung interessierten Personen statt. Sie sind üblicherweise archiviert, so daß es möglich ist, vorangegangene Kommunikation, die auf der Liste stattgefunden hat, einzusehen und nachzulesen. Im Fall von größeren Projekten oder solchen, die auf die Entwicklung von viel genutzten Programmen zielen, existieren mehrere Mailinglisten mit unterschiedlichen thematischen Schwerpunkten. Eine wichtige Differenzierung bildet dabei die Unterscheidung von Entwickler- und Anwendermailinglisten; häufig existiert daneben noch eine Announcement-Liste, auf der wichtige Veränderungen oder die Veröffentlichung von neuen Programmversionen bekanntgegeben werden.

Eine FTP-Seite: Auf dieser Seite können die offiziellen Programmversionen des jeweiligen Projekts heruntergeladen werden.

Ein Concurrent versions system (CVS): Hierbei handelt es sich um ein automatisches Archivierungssystem, mit dem die Beiträge von unterschiedlichen Personen koordiniert werden. Das System dient dazu, gleichzeitig stattfindende Veränderung einer Programmdatei zu verhindern und damit Inkompatibilitäten zwischen den Veränderungen zu vermeiden. Gleichzeitig dient es zur Archivierung von Entwicklungsversionen des Programms und vereinfacht die Wiederherstellung von älteren Programmversionen (Purdy 2001: 6ff.).

Zum anderen zeichnen sich freie Softwareentwicklungsprojekte dadurch aus, daß sie nicht von einem Unternehmen geplant, kontrolliert und organisiert, sondern von einer Vielzahl unterschiedlicher Akteure vorangetrieben werden, die eben nicht gemeinsam Mitglieder einer Organisation sind. Dementsprechend fehlt es den Projekten an den Merkmalen, die für formale Organisationen typisch sind. Dies ist erstens eine formale Hierarchie, die in Organisationen eine Koordination des Handelns der Organisationsmitglieder auf ein Organisationsziel bewirkt (vgl. z.B. March/Simon 1958; Silverman 1970; Scott 1986) und die dauerhafte Verfolgung desselben ermöglichen (vgl. Abels 2001).

Zweitens verfügen die freien Softwareentwicklungsprojekte nicht in demselben Sinn wie Organisationen über ‚Mitgliedschaft' als eine Sonderrolle, die auf Zulassungskriterien basiert (vgl. Luhmann 1964: 35)

und mit denen sie ihre Grenzen zur Umwelt aufrechterhalten und regulieren könnten: Prinzipiell jedermann kann die Projektseiten in Augenschein nehmen, Bugreports verfassen oder Beiträge an die Mailingliste senden – ohne die Liste selbst abonniert zu haben. Jeder kann sich in die Mailingliste selbst ein- und austragen; die Entscheidung über ‚Mitgliedschaft' wird von der betreffenden Person und nicht vom Projekt getroffen. Wie diese erste oberflächliche Charakterisierung von offenen Programmierprojekten zeigt, handelt es sich um eine Form der sozialen Struktur, die soziologische Beobachter hellhörig werden läßt: Es stellt sich vor dem Hintergrund des Fehlens der von Organisationen her bekannten Merkmale die Frage, wie eine Handlungskoordination auf ein gemeinsames Ziel hin gelingt. Damit sind wir mit unserer begrifflichen Klärung am Ende angelangt, die uns einen Problemaufriß beschert und es uns ermöglicht, diesen anschließend in Form der Fragestellung der Untersuchung zu konkretisieren.

Fragestellung der Untersuchung

Im folgenden soll die leitende Fragestellung der Untersuchung konkretisiert und in der Forschungslandschaft verortet werden. Mit der Analyse der sozialen Strukturen, in denen sich die Entwicklung von Technik vollzieht, beschäftigt sich das Feld der Technikgeneseforschung bzw. der Innovationsforschung.[1] In einem ersten Schritt sollen ein knapper Überblick über dieses Feld gegeben, die Entwicklung von Untersuchungsschwerpunkten skizziert und die für das Feld typischen Problemstellungen dargestellt werden. Daran anschließend werden wir uns dem Stand der Forschung im Bereich der Softwareentwicklung zuwenden und zeigen, daß trotz der omnipräsenten Charakterisierung von Software als Zukunftstechnologie nur wenige Untersuchungen vorliegen, die sich mit dem Entwicklungsprozeß dieser Technik beschäftigen. In einem dritten Schritt kommen wir dann zur Konkretisierung des Ziels und des Erkenntnisinteresses der Arbeit. Dieses liegt in der Analyse der Bedingungen und Mechanismen, die eine Entwicklung von freier Software ermöglichen. Die Untersuchung zielt also auf die Aufklärung der Organisation des Technikentwicklungsprozesses im Bereich der freien Software. An diesem Punkt werden wir die übergeordnete Fragestellung mithilfe von zwei untersuchungsleitenden Teilfragestellungen spezifizieren.

1 Die Begriffe Innovation(-sforschung) und Technikgenese(-forschung) werden im folgenden synonym gebraucht. Zwar resultieren die unterschiedlichen Bezeichnungen aus verschiedenen disziplinären Traditionen resp. der Wirtschafts- beziehungsweise der Sozialwissenschaften, mittlerweile sind die beiden Felder jedoch kaum noch zu trennen, da die Theorieangebote und die empirischen Untersuchungen von den beteiligten Disziplinen wechselseitig rezipiert werden.

Einordnung: Das Feld der Technikgeneseforschung

Die Technikforschung interessierte sich bis in die 1980er Jahre hinein vor allen Dingen für die positiven und negativen Folgen der Einführung und Anwendung von Technik. Der soziologisch orientierten Technikforschung kam dabei vor allem die Aufgabe zu, nach den gesellschaftlichen Auswirkungen zu fragen, wobei hier wiederum der Bereich ‚Technik und Arbeit' den dominanten Fokus der Forschung darstellte, während das Thema (groß-)technologische Risiken erst Anfang der 1980er Jahre mit einer Etablierung der Risikosoziologie zu einem prominenten Forschungsgegenstand wurde.[2] Die starke Folgenorientierung ist in der Retrospektive vor allen Dingen zwei Ursachen geschuldet: Zum einen existierte (und existiert) ein von Seiten der Politik an die Wissenschaft herangetragener Bedarf nach wissenschaftlicher Expertise zur Steuerung der technischen Entwicklung. Von Interesse war und ist dabei die Gewinnung von sicherem Wissen über den (zukünftigen) Einsatz von (neuen) Techniken. Diesem Bedarf wurde durch die Institutionalisierung der Technikfolgenabschätzung (TA) und der Schaffung entsprechender Forschungseinrichtungen entsprochen.[3] Neben diesen wissenschaftsexternen, exogenen Ursachen resultierte die Ausrichtung der soziologischen Technikforschung auch aus fachimmanenten Gründen: Bis in die 1980er Jahre hinein wurde die soziologische Techniktheorie von unterschiedlichen Spielarten des technischen Determinismus sowie monologischen Erklärungsmustern der Technikentwicklung dominiert, die eine genauere Analyse der Entstehungs- und Entwicklungsbedingungen als unattraktiv und wenig erkenntnisträchtig erscheinen ließ.

Die starke These eines technischen Determinismus setzt sich aus zwei Grundannahmen zusammen: Zum einen wird davon ausgegangen, daß die Entwicklung von Technik eine Eigendynamik besitzt, also einer inhärenten Logik folgt.[4] Mit dieser Annahme wird die Entwicklung von Technik aus dem Bereich des Sozialen herauskatapultiert. Zum anderen wird ein konsequentieller Determinismus postuliert, der von einer Do-

2 Eine Übersicht über die Entstehung des Forschungsfeldes Risikosoziologie findet sich bei Krücken 1994. Während sich das soziologische Interesse im englischen Sprachraum bereits zu Beginn der 1980er Jahre dem Thema Risiko zuwandte (vgl. Douglas/Wildavsky 1982; Wynne 1982; Perrow 1984), fand eine Beschäftigung innerhalb der deutschsprachigen Soziologie erst im Anschluß an die Monographien von Luhmann (1986) und Beck (1988) statt.

3 Zur Geschichte der Institutionalisierung der TA siehe ausführlich den Sammelband von Bechmann 1996.

4 Hier sicherlich am prominentesten Schelsky 1965.

minanz der technischen Entwicklung gegenüber anderen (sozialen) Faktoren ausgeht. Sozialer Wandel kommt demnach u.a. durch Entwicklungen im Bereich der als extern und ‚nicht sozial' konzipierten Technik zustande, während der Bereich des Sozialen diese Veränderungsprozesse passiv erleidet.[5] Monologische Erklärungen der technischen Entwicklung verweisen hingegen an Stelle einer technischen Eigendynamik auf eine Strukturlogik, die ihrerseits den technischen Wandel bestimmt bzw. prägt. Hier seien vor allen Dingen die Logik der Herrschaft (Marcuse 1967), die Logik der ökonomischen Verwertung (Marx 1969), kulturelle Leitbilder (Mumford 1934) sowie die anthropologische Erklärungen (Gehlen 1957) genannt.[6] Allein schon die Konkurrenz der Ansätze, die jeweils versuchen, sämtliche technische Wandlungsprozesse auf eine Strukturlogik zurückzuführen, relativiert den jeweiligen Erklärungsanspruch.

Diese kurze Skizze der Theorielandschaft macht deutlich, daß sich das Forschungsfeld der Technikgenese nicht nur durch die Etablierung eines neuen Forschungsschwerpunktes innerhalb der sozialwissenschaftlichen Technikforschung gegenüber einer starken Folgenorientierung zu emanzipieren hatte, sondern auch neue Theorieangebote entwickeln mußte, um den Weg für die Analyse von Technikentwicklungsprozessen frei zu machen. In den 1980er Jahren wandten sich die Forschungsaktivitäten dann den Entstehungsbedingungen von Technik zu. Diese Verschiebung entsprang zum einen dem Niedergang der großen monologischen Entwicklungstheorien und zum anderen aus dem Bedarf der Technikfolgenabschätzung nach Wissen bezüglich der Eingriffsmöglichkeiten für die Steuerung von Technik. Der Wissensbedarf wurde als ‚Steuerungsdilemma' (Collingridge 1981) formuliert und kulminiert in der Frage nach dem ‚richtigen' Zeitpunkt für eine Steuerung von Technik und der Initiierung einer Technikfolgenabschätzung: Während die Steuerung von Technik zu einem frühen Zeitpunkt aufgrund des unsicheren Wissens über mögliche Folgen kaum möglich oder zumindest mit einem hohen Risiko behaftet ist, liegt das Problem der Techniksteuerung zu einem späten Zeitpunkt in der ‚Verhärtung' der Technik durch die Pfadabhängigkeit der getroffenen Entscheidungen. In der Konsequenz führte dieses Dilemma dann zu einem Anwachsen des Interesses an Prozessen der Technikgenese, mit dem Ziel, genauere Kenntnisse über die Entwicklungsbedingungen von Technik zu erlangen, um dort steuernd einzugreifen.[7]

5 Z.B. deutlich in der These des ‚cultural lag' von Ogburn 1966: 200 ff.

6 Zusammenfassend auch Rammert 1993: 154 ff.

7 Zum Zusammenhang von Technikfolgenabschätzung und Technikgeneseforschung siehe Rammert 1994: 17 ff.; Dierkes/Marz 1994: 37.

Die sich Mitte der 1980er Jahre formierende Technikgeneseforschung beschäftigt sich nun mit den Entstehungsprozessen von Technik (Rammert 1993: 19). Im Unterschied zur älteren folgenorientierten Forschung wird der Erfolg von Technik nicht zum Ausgangspunkt für weitere Untersuchungen gemacht, sondern selbst als erklärungsbedürftig wahrgenommen, wie es pointiert in der Formulierung von Bijker und Pinch zum Ausdruck kommt: „The success of an artefact is precisely what needs to be explained. For a social theory of technology it should be the explanandum not the explanans.“ (Bjiker/Pinch 1984: 406)[8] Die Technikgeneseforschung möchte demnach den Anspruch auf eine gesellschaftlich geprägte Technik einlösen, der im Zusammenhang des Verständnisses von ‚Technik als sozialen Prozeß‘ immer wieder erhoben wird (Rammert 1994: 15). Dabei wird angenommen, daß die Entwicklung von Technik eine soziale Betätigung darstellt, die durch eine Vielzahl von Faktoren beeinflußt wird. Das Ziel der Technikgeneseforschung besteht nun darin, den bis dato als ‚black box‘ behandelten Prozeß zu untersuchen und die sozialen Faktoren zu ermitteln und zu analysieren, die für eine Gestaltung der Technik verantwortlich sind. In Abgrenzung zu den unterschiedlichen Formen des technischen Determinismus wird von einer prinzipiellen Verlaufsoffenheit und Gestaltbarkeit ausgegangen.[9] Zweck-Mittel-Zusammenhänge werden demnach nicht ‚entdeckt‘ und in einer bestmöglichen, eventuell noch zu optimierenden Form vergegenständlicht, sondern stellen vielmehr Ergebnis sozialer Konstruktionsarbeit dar.

Im Unterschied zu eindimensionalen Erklärungsmodellen wird davon ausgegangen, die technische Entwicklung sei nicht durch eine dominante Strukturlogik bestimmt, sondern durchlaufe unterschiedliche gesellschaftliche Teilbereiche (Rammert 1993: 162), wobei eine Vielzahl von heterogenen Akteuren oder Akteursgruppen auf den Prozeß Einfluß nehmen (Bijker/Pinch 1984: 414). Damit sind zwei unterschiedliche Erklärungsebenen angesprochen, auf denen sich die Konzepte der Technikgenese bewegen: Zum einen finden sich Konzepte wie ‚Social Construction of Technologie‘ (SCOT),[10] die auf Akteursebene ansetzen und Technikentwicklung durch die Interaktion von Akteuren erklären,

8 Vgl. auch zusammenfassend Hack 1995: 19.

9 Dies wird z.B. durch die folgende Bemerkung zur Gestaltung von Technik deutlich: „They might have been otherwise: this is the key to our interest and concern with technology. Technologies do not, we suggest, evolve under the impetus of some necessary inner technological or scientific logic.“ (Bijker/Law 1992: 3)

10 Eine Darstellung der unterschiedlichen im Bereich der Technikgeneseforschung vorfindlichen Theorien würde den Rahmen sprengen. Verwiesen sei hier auf Bijker/Pinch 1984; Bijker/Law 1992; Bijker 1987, 1995.

die verschiedene von der Technik zu lösende Problemkonzeptionen vertreten, zum anderen Ansätze, die sich auf eher makrosoziologische Kategorien wie Geschlecht, die Logik der Ökonomie oder die des Militärs stützen, Technikentwicklung in unterschiedlichen gesellschaftlichen Teilbereichen verorten und so den Anschluß an die soziologische Gesellschaftstheorie suchen (z.B. „Social Shaping of Technology" – SST).[11]

Seit Anfang der 1990er Jahre läßt sich eine Verschiebung des Forschungsinteresses in Richtung der sozialen Organisation von Technikgenese- bzw. Innovationsprozessen beobachten. Im Mittelpunkt steht hier die Frage, wie Ziele gesetzt, Strategien verfolgt und unter welchen Rahmenbedingungen Beträge zu Technisierungsprojekten geleistet werden. Damit aufgeworfen wird die Frage nach der Koordination von sozialem Handeln auf ein Ziel hin und nach den Ansprüchen an die sozialen Strukturen, die eine solche Koordination leisten.

Softwaregenese: Stand der Forschung

Die Technikgeneseforschung ist mittlerweile ein ausdifferenziertes Forschungsfeld, in dem für eine Vielzahl von Technikfeldern aktuelle und historische Fallstudien vorliegen. Auch die Informations- und Kommunikationstechnologie ist zum Gegenstand dieses Forschungsbereichs geworden, wobei hier zu unterscheiden ist zwischen Untersuchungen, die sich auf die Hardwareentwicklung beziehen, und solchen, welche die Entwicklung von Software zum Gegenstand haben. Im Bereich der Hardware finden sich Studien zur Entstehung der ersten Computer (Stach 2001), der PCs (Weyer et al. 1997), Aneignungsformen des Computers im Alltag (Rammert 1991) sowie historische Überblicksdarstellungen (z.B. Ceruzzi 2000, Rode/Hansen 1993). Mit der Entwicklung der Formalisierung beschäftigen sich Krämer (1988) und Heintz (1993), wobei diese Arbeiten eher ideen- oder disziplinenhistorisch einzuordnen und weniger dem engeren Bereich der Technikgeneseforschung zuzurechnen sind.

Im Vergleich zur gut aufgearbeiteten Technikgenese im Bereich der Computer-Hardware finden sich im Bereich der Softwareentwicklung nur wenige soziologisch orientierte Arbeiten, die sich explizit mit Innovationsprozessen beschäftigen. In ein weites Verständnis dieses Feldes fallen Gesamtdarstellungen zur Entwicklung der Softwareindustrie (z.B. Campbell-Kelly 1995), Forschungen zur Entwicklung von Program-

11 Vgl. die beiden Sammelbände von McKenzie/Wajcman 1985, 1999.

miersprachen (Stach 2001) sowie die im Kontext des interdisziplinären Forschungsprojekts ‚Sozialgeschichte der Informatik' an der FU Berlin entstandenen Arbeiten zu Handlungsformen in der Informatik (Braun 1998), der Bedeutung von Metaphern in der Entwicklung der Disziplin (Busch 1998) und der Rekonstruktion von Analogien zwischen Mensch und Maschine im Denken der Informatiker (Stach 1998). Auch wenn die letztgenannten Arbeiten m.E. zum Verständnis von Innovationsprozessen beitragen, liegt ihr gemeinsamer Fokus in ihrem disziplinenhistorischen Zugang.[12] Unter dem Schlagwort ‚Softwarekrise' beschäftigt sich der Beitrag von Theißling (1995) mit dem Zusammenhang zwischen Softwareentwicklungspraxis und Qualitätsproblemen, wobei dieser Publikation zwar die Analyse einschlägiger Literatur, nicht aber die Beobachtung des Softwareentwicklungsprozesses selbst zugrunde liegt.

Enger auf Innovationsprozesse im Bereich der Software fokussiert und eindeutig dem Bereich der Technikgeneseforschung zuzuordnen ist die mittlerweile klassisch zu nennende Monographie von Weltz/Ortmann ‚Das Software-Projekt', die auf einer Untersuchung von 46 Softwareprojekten basiert (1992). Die quantitativ angelegte Studie widmet sich vorrangig der ‚organisatorischen Seite der Projektabwicklung' (ebd. 11) und legt einen Schwerpunkt auf die praktische Wirksamkeit normativer Modelle der Softwareentwicklung, wie man sie in der Literatur zum Software Engineering findet. Zentrales Ergebnis ist, daß die Entwicklungspraxis vom planhaften Vorgehen der Modelle erheblich abweicht (ebd. 52), wobei dies für sämtliche Phasen der Softwareentwicklung gilt. Als besonders problematisch heben die Autoren den häufig späten Zeitpunkt einer Anwenderbeteiligung hervor, an dem bereits wichtige Entscheidungen getroffen sind (ebd. 73), sowie eine häufig erst im Verlauf des Entwicklungsvorhabens entstehende Projektorganisation (ebd. 51). Vor diesem Hintergrund wundert es dann auch wenig, daß die Autoren zusammenfassend von einer nach wie vor hohen Aktualität des Themas ‚Softwarekrise' sprechen, die sich insbesondere in überzogenen Termin- und Kostenbudgets und einem hohen Anteil an nicht eingesetzter oder nicht einsatztauglicher Software zeigt (ebd. 158). Zur Bewältigung dieser Situation fordern die Autoren eine stärkere Orientierung am Bild des dynamischen Projektmanagements, anstatt an vorgegebenen Plänen festzuhalten (ebd. 176).

Während sich die Arbeit von Weltz und Ortmann hauptsächlich auf organisatorische Aspekte der Softwareentwicklung bezieht, werden innerhalb des von Brodbeck und Frese (1994) herausgegebenen Sammel-

12 Vgl. auch die professionshistorische Perspektive bei Schulz-Schaeffer 1996.

bands eher psychologische Dimensionen der Programmierarbeit thematisiert. Die unterschiedlichen Beiträge entstammen dem quantitativ orientierten und durch Leitfadeninterviews unterstützten Forschungsprojekt IPAS,[13] das sich selbst nicht im Feld der Technikgeneseforschung verortet, jedoch einige Aspekte beinhaltet, die für dieses Forschungsgebiet einschlägig sind: So wird eine Tätigkeitsanalyse durchgeführt mit dem Ziel, Schwerpunkte und Kompetenzanforderungen zu ermitteln (Brodbeck 1994), die Vorteile und Schwierigkeiten der Beteiligung von Anwendern bei der Entwicklung von Software untersucht (Heinbokel 1994) und der Stellenwert unterschiedlicher Qualifikationsprofile von Spitzenkräften für den Erfolg der Entwicklungsprojekte einer Analyse unterzogen (Sonnentag 1994). Als Ergebnis kann festgehalten werden, daß sich die Zusammenarbeit zwischen Entwicklern und Anwendern in vielen Fällen als problematisch gestaltet, was sich insbesondere in einer häufigeren Revision der Aufgabendefinition, einer geringeren Qualität des Produkts und einer niedrigen Erfolgsquote zeigt. Als Ursachen dafür werden komplexere Problemstellungen, ein Ansteigen der Komplexität von Kommunikationsprozessen, ein schlechter Informationsfluß, unklare Zuteilungen von Kompetenzen, Motivationsprobleme der Benutzer und ein höheres Maß an Interessendivergenz im Fall von Projekten mit Benutzerbeteiligung genannt (Heinbokel 1994: 119 ff.). In bezug auf die Kompetenzanforderungen des Feldes stellt Brodbeck fest, die hohe Interdependenz von Teilaufgaben führe zu hohe Kommunikations- und Lernanforderungen. Die Anforderungen der Entwicklungspraxis differieren dabei z.T. erheblich mit den in der Ausbildung erworbenen Kompetenzen, wobei diese Diskrepanz vermutlich nicht durch den Einsatz von neueren Entwicklungswerkzeugen verringert wird (ebd.: 24). In bezug auf den Erfolg von Entwicklungsprojekten läßt sich feststellen, daß die Spitzenkräfte hier eine Schlüsselstellung einnehmen (Sonnentag 1994: 35), und nicht nur die fachliche Kompetenz, sondern – für Soziologen wenig überraschend – auch Kooperations- und Kommunikationsfähigkeit von zentraler Bedeutung sind. Diese Anforderungen sollten daher sowohl innerhalb der Ausbildung als auch in der berufsbegleitenden Weiterbildung stärker berücksichtigt werden (ebd. 49).

Neueren Datums ist die Monographie „Innovationen in der Softwareindustrie“ von Konrad und Paul (1999). Sie basiert auf einer empirischen Untersuchung von jeweils drei Fallstudien zur Entwicklung von betriebswirtschaftlicher Standardsoftware und Prozeßleitsystemen. Die Schwerpunkte der komparativ angelegten Untersuchung liegen dabei auf

13 IPAS: Interdisziplinäres Projekt zur Arbeitssituation in der Softwareentwicklung.

den organisationalen Strukturen und Marktstrategien der beteiligten Unternehmen sowie auf der Arbeitssituation von Entwicklern. Ziel ist die Identifikation von spezifischen Innovationsmustern, wobei darunter in Anschluß an Hirsch-Kreinsen (1997) differierende Entwicklungspfade verstanden werden, die sich aus einem Bündel von kulturellen, organisatorischen, institutionellen, politischen und sozioökonomischen Faktoren zusammensetzen (Konrad/Paul 1999: 31). In der Monographie findet sich eine ausführliche Beschreibung der Unternehmen, der Unternehmensgeschichte sowie der Unternehmenskooperation, die sich allerdings nahezu ausschließlich auf die formalen Merkmale beschränkt. Als zentrales Ergebnis der Untersuchung läßt sich festhalten, daß in beiden untersuchten Bereichen Veränderungen des Innovationsmusters erkennbar sind: Für das Feld der betrieblichen Standardsoftware läßt sich eine Tendenz zu überbetrieblichen Kooperationen zwischen den Herstellern feststellen (ebd. 214), während im Bereich der Prozeßleitsysteme Konzentrationstendenzen zu beobachten sind (ebd. 219). Innovative Softwareentwicklung resultiert dabei vor allen Dingen aus informationstechnischen Veränderungen, wie z.B. der Einführung von einer neuen Hardwaregeneration (ebd. 86; 114) oder veränderten Anforderungsprofilen (z.B. der Einführung von grafischen Benutzeroberflächen; ebd. 87). Insgesamt bleibt die Untersuchung an den oberflächlichen Merkmalen der Projekte verhaftet und unternimmt nicht den Versuch, diese weiter zu analysieren. So wird beispielsweise das Scheitern und Gelingen von Kooperationen zwar konstatiert, jedoch wird den Fragen nach den Ursachen, verantwortlichen strukturellen Merkmalen und Erfolgsbedingungen nur sehr eingeschränkt nachgegangen.

Sieber und Henninger (2001) widmen ihre qualitative Untersuchung der Softwareentwicklung innerhalb von kleinen Unternehmen und interessieren sich vor allen Dingen für die Umsetzung von Vorgehensmodellen des Software Engineering in der Entwicklungspraxis. Zentrale Erkenntnis der Studie ist ebenso wie bei Weltz und Ortmann auch hier, daß zwischen der Entwicklungspraxis und den normativen Modellen große Differenzen bestehen. Methodisch gewendet sei daher eine ‚realitätsnahe' Beschreibung des Prozesses nur unter der Berücksichtigung von Modellen der Softwareentwicklung, Projekttypen und Arbeitspraktiken möglich (Sieber/Henninger 2001: 12). Während sie in bezug auf die Projekttypen zwischen Vertragsentwicklung, Produktentwicklung und In-Haus-Entwicklung unterscheiden (ebd. 7), kann zwischen erfahrungs- und theoriegeleiteten Arbeitsstilen differenziert werden (ebd. 11). Softwareentwicklungsmodelle lassen sich hingegen entweder einer Management-, Entwicklungs- oder einer Anwendungsperspektive zuordnen. Mit Hilfe dieser Unterscheidungen ermitteln die Autorinnen günstige

oder weniger erfolgversprechende Merkmalskombinationen innerhalb von Entwicklungsprojekten. Als defizitär beschreiben sie die Nutzerpartizipation, da in keinem der untersuchten Unternehmen eine systematische Beteiligung von Anwendern stattfindet.

Als einschlägig für den Bereich der *freien Softwareentwicklung* kann die innerhalb des Projektes ‚Kulturraum Internet'[14] durchgeführte qualitative Untersuchung der ‚Internet Engineering Task Force' (IETF) gelten. Untersuchungsgegenstand bildet hier die Mailingliste des Projekts, auf der während des Untersuchungszeitraums eine neue Version des Internet-Protokolls TCP/IP entwickelt wird, also eine technische Innovation, die unter den Rahmenbedingungen einer freiwilligen und selbstselektiven Mitarbeit unter Zuhilfenahme von computervermittelter Kommunikation hervorgebracht wird (Hofmann 1999). Die Untersuchung zeichnet den Aufstieg und Fall einer technischen Innovation nach und widmet sich vor allen Dingen der Analyse des Kommunikationsprozesses auf der Mailingliste und dem Selbstverständnis der beteiligten Akteure. Als besonders augenfällig erweist sich der „kooperative Arbeitsstil" und die „konzeptionelle Aufgeschlossenheit" (Hofmann 1999: 187), mit der die Entwickler sich um ein Verständnis der existierenden technischen Lösungen und der Diskussion von innovativen Vorschlägen bemühen.

Neben dieser Fallstudie ist in der jüngeren Vergangenheit ein größerer Korpus von Literatur zu freier Software entstanden, der zwar für die Erklärung der Genese von freier Software von Bedeutung ist, sich jedoch nicht unmittelbar dem Feld der Technikgeneseforschung zurechnen läßt. Hierzu zählen allgemeine Überblicksarbeiten zu freier Software (z.B. Grassmuck 2002; Siekmann 2000), Untersuchungen zu den Motiven der Beteiligten (z.B. Lerner/Tirole 2002; Butler et al. 2002; Osterloh et al. o.J. Browne 1998) und zu freien Softwarelizenzen (Healy/Schussman 2003; Bonaccorsi/Rossi 2002), quantitative Studien zur Größe von Projekten und den Beiträgen unterschiedlicher Entwickler (z.B. Ghosh/Robles/Glott 2002) sowie Analysen der Beiträge von Firmen zur Entwicklung der Software (z.B. Wichmann 2002). Daneben finden sich Untersuchungen, in denen unterschiedliche Elementen des Entwicklungsprozesses im Fokus stehen, wobei hierunter die Rolle der Benutzer (v.Hippel 2001; Franke/v.Hippel 2003) und die Technik der Modularisierung des Programmquellcodes für die Entwicklung von freier Software (Narduzzo/Rossi 2003) fallen.

14 Der Endbericht des Projektes findet sich als Online-Publikation unter: http://duplox.wz-berlin.de.

Die Literaturübersicht zeigt, daß einige Anstrengungen unternommen wurden, das Feld der Softwareentwicklung einer sozialwissenschaftlichen Analyse zugänglich zu machen. Allerdings kontrastieren die wenigen empirischen Fallstudien dann doch recht stark mit der immer wieder betonten hohen Relevanz von Informations- und Kommunikationstechnologien. Fortgeschrittener ist, zumindest von der Anzahl der Untersuchungen her betrachtet, der Stand der Forschung im Bereich der freien Software: Die in der jüngeren Vergangenheit publizierten Untersuchungen thematisieren wichtige Aspekte dieser Form der Technikentwicklung und sind für die vorliegende Arbeit von Bedeutung. Trotz dieser wichtigen Schritte stehen bislang noch Beiträge aus, die von den einzelnen empirischen Ergebnissen einen Brückenschlag zu den Befunden der soziologischen Technikgenese- und Innovationsforschung unternehmen und das Feld der freien Softwareentwicklung systematisch fruchtbar machen.

Erkenntnisinteresse und Fragestellung der Untersuchung

Ausgangspunkt der Untersuchung bildet die Annahme, daß die Entwicklung von freier Software in neuartiger Weise organisiert ist. Diese These gründet sich auf drei Beobachtungen hinsichtlich der Rahmenbedingungen, denen die Entwicklung unterliegt.

Erstens findet sie in einem Netzwerk heterogener Akteuren statt, zu denen, wie weiter oben dargestellt, Core-Entwickler, sporadische Entwickler, Pilot- und Normalanwender sowie Distributoren zählen. Es handelt sich also um ein Netzwerk, das sich aus Personen und Organisationen zusammensetzt. Aufgrund des jedermann durch freie Softwarelizenzen gewährten Modifikationsrechts handelt es sich um ein *nicht-exklusives Netzwerk*: Keiner der Akteure ist in der Lage, den Kreis der Beteiligten zu beschränken, den Zugang zu kontrollieren und beteiligungswillige Personen oder Organisationen von der Entwicklung auszuschließen oder von eigenen Entwicklungsbeiträgen abzuhalten. Die prinzipielle Offenheit des Entwicklungs-Netzwerks zeigt sich darin, daß jeder, der über die entsprechenden technischen Kompetenzen verfügt, an die bislang geleistete Entwicklungsarbeit anderer Programmierer anschließen und eigene Beiträge verfassen kann, allerdings nur unter der Prämisse, daß die modifizierten Versionen des Programms wiederum anderen Personen zugänglich gemacht werden.[15] Hierin unterscheidet

15 Das gilt zumindest für den Fall der sogenannten Copyleft-Lizenzen.

sich der Forschungsgegenstand von der Technikentwicklung innerhalb von Organisationen oder interorganisationalen Kooperationen, bei der der Zugang zum Innovationsprozeß üblicherweise restriktiv gehandhabt wird.

Zweitens sorgt das jedermann gewährte Recht zur Modifikation dafür, daß keiner der beteiligten Akteure über eine durch die Softwarelizenz rechtlich abgesicherte zentrale koordinierende und den Softwareentwicklungsprozeß bestimmende Rolle verfügt. Unabhängig von bislang geleisteten Beträgen und den Beiträgen anderer Personen steht es jedem frei, einen eigenen Beitrag zu verfassen, auch wenn dieser nicht den Intentionen des ursprünglichen Programmautors oder anderen Personen entspricht oder sogar entgegen deren Willen erfolgt.

Drittens findet die Koordination der Entwicklungsarbeit innerhalb von Softwareentwicklungsprojekten nahezu ausschließlich mithilfe digitalisierter, computervermittelter Kommunikation statt. Die Entwickler von freier Software können nur in Ausnahmefällen – wie beispielsweise im Fall von Konferenzen, Tagungen oder Messen – mit ihren Kooperationspartnern innerhalb von face-to-face-Situationen kommunizieren. Entwicklungsbezogene Kommunikation und Entscheidungsprozesse finden überwiegend auf öffentlich zugänglichen Mailinglisten, also vermittels asynchroner, textbasierter, digitaler Medien statt, wodurch weit entfernte Personen kooperieren und gemeinsam das Entwicklungsprojekt vorantreiben können. Daher läßt sich auch von einer Enträumlichung des Entwicklungsprozesses sprechen. Zwar werden auch anderen Innovationsprozesse in vielfältiger Weise durch Computer, Computernetzwerke und digitale Medien unterstützt, neuartig an der freien Softwareentwicklung ist allerdings die Ausschließlichkeit oder zumindest die stark dominante Rolle, die computervermittelte Kommunikation hier spielt. Vor dem Hintergrund der knapp skizzierten Rahmenbedingungen läßt sich das allgemeine Erkenntnisinteresse wie folgt zusammenfassen:

Wir zielen mit der Untersuchung des Entwicklungsnetzwerks auf die Beschreibung und Analyse der sozialen Strukturen, in denen sich die Entwicklung von freier Software vollzieht. Im Zentrum des Interesses steht dabei die Frage, wie die Entwicklung von freier Software organisiert ist, oder, präziser formuliert: welche soziale Bedingungen und Mechanismen die Entwicklung von freier Software ermöglichen. Ein wichtiger Aspekt der Beobachtung liegt dabei auf dem Zusammenhang zwischen den sozialen Strukturen einerseits und der Technikgestaltung andererseits. Damit ist die Frage nach der Folgenhaftigkeit der sozialen Strukturen auf die Formung und Gestaltung von Technik angesprochen.

Dieses allgemeine, der Untersuchung übergeordnete Erkenntnisinteresse läßt sich durch zwei leitende Fragekomplexe näher spezifizieren:

Erstens ist die Existenz des Netzwerks freier Softwareentwicklung, welches sich in freie Softwareprojekte mit den oben weiter ausgeführten Eigenschaften von Nichtexklusivität der Teilnehmerschaft, Polyzentrizität und digitaler Medienbasiertheit untergliedert, erklärungsbedürftig. Damit ist die Frage nach dem Entstehen des Entwicklungsnetzwerks aufgeworfen. Die erste untersuchungsleitende Frage lautet daher:

Wie läßt sich das Entstehen der sozialen Strukturen, in denen sich die Entwicklung von freier Software vollzieht, erklären und welche Bedingungen haben hierauf Einfluß genommen?

Zweitens wird in der Literatur angenommen, die Entwicklung von Technik sei auf Organisationen angewiesen, die sowohl für die Mobilisierung der notwendigen Ressourcen als auch für die Koordination des Handelns von unterschiedlichen Akteuren auf ein gemeinsam verfolgtes Entwicklungsziel hin sorgen. Vor dem Hintergrund der Eigenschaften des Netzwerks von Nichtexklusivität der Teilnehmerschaft und Polyzentrizität der Entscheidungsstruktur stellt sich die Frage, welche sozialen Mechanismen Kooperation ermöglichen, eine Koordination des Handelns der Beteiligten leisten und wie Ressourcen für die Entwicklung mobilisiert werden. Dieser zweite untersuchungsleitende Fragenkomplex läßt sich wie folgt zusammenfassen:

Wie sind freie Softwareentwicklungsprojekte organisiert, welche institutionalisierten Regeln und Rollendifferenzierungen existieren, die eine Entscheidungsfindung ermöglichen und kollektives Handeln auf ein Entwicklungsziel hin koordinieren?

Theoretische Ausgangspunkte: Das Konzept der Innovationsnetzwerke

An dieser Stelle soll nun der Rahmen entwickelt werden, mit dem die Technikentwicklung durch die Kooperation unterschiedlicher Akteure im Fall von freien Softwareentwicklungsprojekten beschrieben und eingefangen werden kann. Theoretischer Bezugspunkt der Arbeit bildet das Konzept der Innovationsnetzwerke – eine Theorie, die entwickelt wurde, um die Kooperation heterogener Akteure bei der Erzeugung von innovationsrelevantem Wissen zu analysieren. Es wurde an Fallstudien der Technikgenese entwickelt, bei denen *interorganisationale* Kooperationsbeziehungen eine wesentliche Rolle spielen und bietet sich aus mehreren Gründen auch für die Analyse des *personalen* Netzwerks von freien Programmierprojekten an:

Erstens stellt das Konzept die Frage nach der Konstitutionslogik solcher Netzwerke[1] und offeriert im Anschluß an die Transaktionskostenökonomie eine Antwort auf diese Frage. Zweitens verwirft das Konzept die Vorstellung einer linearen Technikentwicklung und geht statt dessen davon aus, daß es sich hierbei um einen mehrfach rückgekoppelten Prozeß handelt – eine Struktur, wie sie auch im Fall der freien Softwareentwicklung vorzufinden ist. Drittens stellt es einen Rahmen zur Analyse der Erzeugung von innovationsrelevantem Wissen bereit: Im Mittelpunkt stehen Lernprozesse, die durch das Zusammentreffen der Wissensbestände von Herstellern und Anwendern in Gang gesetzt werden, sowie die Voraussetzungen, die für solche Kooperationen erfüllt werden

1 Eine Frage, die in der Literatur zu sozialen Netzwerken bislang unterbelichtet geblieben ist (vgl. Rölle/Blättel-Mink 1998).

müssen. Viertens spielt das Verhältnis zwischen einer vertraglichen Regulierung und einer Vertrauensbasierung von Netzwerken eine prominente Rolle, das, wie wir weiter unten sehen werden, auch in unserem Fall von Bedeutung ist.

Markt – Hierarchie – Netzwerk

Eine erste begriffliche Einordnung von freien Softwareentwicklungsprojekten hatten wir bei der Darstellung des Forschungsgegenstands unternommen, indem wir diesen als Netzwerk bezeichnet hatten. Allerdings hatten wir die Bedeutung dieser begrifflichen Bestimmung nicht erläutert, sondern den Begriff provisorisch eingeführt. Die Auseinandersetzung mit dem Begriff ‚Netzwerk' soll nun nachgeholt werden.

Geht es – wie in der vorliegenden Untersuchung – um die Koordination von sozialem Handeln, werden üblicherweise drei Typen von sozialen Strukturen ins Spiel gebracht: Dies sind ‚Hierarchie' (oder formale Organisation), ‚Markt' sowie der in der jüngeren Diskussion vielbeachtete Typus ‚Netzwerk' (Weyer 2000a; Powell 1996: 221).[2] Seinen Ursprung hat diese Typologie in der Transaktionskostenökonomie, die auf den frühen Arbeiten von Ronald H. Coase (1937) aufbaut. Grundlegende Annahme der Transaktionskostenökonomie ist, daß wirtschaftliche Transaktionen in unterschiedlichen Formen von sozialen Strukturen – sogenannten Strukturalternativen – abgewickelt werden können, wobei keine die in allen Fällen überlegene Alternative darstellt.

„The transaction cost approach affords a partial answer: there are so many kinds of organizations because transactions differ so greatly and efficiency is realized only if governance structures are tailored to the specific needs of each type of transaction." (Williamson 1981: 568)

Maßgebliche Dimensionen, die der Ansatz zur Analyse von Transaktionen in Anschlag bringt, bilden (a) die Unsicherheit der Marktteilnehmer bezüglich der Marktsituation, (b) die Häufigkeit mit der eine Transakti-

2 Die Bezeichnungen sind in der Literatur nicht einheitlich. Wiesenthal (2000) spricht von Markt, Organisation und Gemeinschaft, Mayntz (1992) von Markt, Organisation und Verhandlungssystem, Willke (2001) schwankt zwischen den Typologien Demokratie/Organisation/Verhandlungssystem und Markt/Hierarchie/Netzwerk. Zu einer grundlegenden Kritik dieser Typologie siehe z.B. Kappelhoff (2000). Dieser versteht in Anschluß an die allgemeine Netzwerkanalyse (siehe z.B. Jansen 2003; Granovetter 1985; Garton/Haythronthwaith/Wellman 1997) Netzwerke als übergreifende Kategorie, in die auch Märkte und Organisationen fallen.

on auftritt, sowie (c) die Höhe der Investitionskosten, die für eine der Organisationsformen getätigt werden muß (Williamson 1981: 555; 1986: 111).

Williamson unterstellt den Teilnehmern einer Transaktion, sie seien bei der Auswahl der Organisationsform vom Ziel einer größtmöglichen Reduktion der Transaktionskosten geleitet (1985: 17, 551; 1986: 187), wobei er hierunter die Kosten für die Anbahnung und Aufrechterhaltung der Transaktion versteht.[3] Im einzelnen sind dies

> „costs of drafting, negotiation, and safeguarding of an agreement […], the maladaption costs incurred when transactions drift out of alignment in relations […], the haggling costs incurred if bilateral efforts are made to correct ex post misalignments […], the setup and running costs associated with the governance structures (often not the courts) to which disputes are referred“ und „the bonding costs of effecting secure commitments.“ (1985: 20 f.).

Von der Transaktionskostenökonomie können nun durch die Analyse von Transaktionen, Ermittlung der Kosten unterschiedlicher Strukturalternativen und Annahmen über die Orientierung der Akteure (Minimierung von Transaktionskosten) Voraussagen über die gewählte Alternative abgeleitet werden. Der entscheidende Schritt, der mit der Transaktionskostenökonomie genommen wird, ist, daß unterschiedliche Formen der Handlungskoordination unter einen gemeinsamen theoretischen Rahmen gefaßt und in bezug auf bestimmte Merkmale verglichen werden. In Betracht gezogen werden von Williamson die Koordination durch die Hierarchie, den Markt und Hybridformen (1996: 171).

Die damit aufgeworfene komparative Perspektive hat sich als fruchtbar für die Diskussion um die gesellschaftliche (Selbst-)Steuerung in der Steuerungstheorie erwiesen (z.B. Willke 2001) und hat die Theorieentwicklung im Bereich der Netzwerkforschung stark inspiriert. Hier liegt auch der Fokus der Diskussion, in der klassische Arbeiten zu individualistischen (Markt) und kollektivistischen (Hierarchie) Modellen von sozialer Ordnung miteinander verglichen und durch die dritte Form des sozialen Netzwerks kontrastiert werden. Die dort prominente Typologie von Koordinationsformen, die im Zuge der Debatte um Strukturalternativen eine soziologische Aufweitung gegenüber der engen, ausschließlich auf Transaktionskosten fokussierten Perspektive erfuhr, soll im folgenden dargestellt werden.[4] Innerhalb der Netzwerkforschung

3 Williamson unterscheidet hier zwischen ex ante- und ex post-Kosten einer Transaktion. Vgl. Williamson 1985: 20 ff.

4 Zwar ist die Transaktionskostenökonomie ein wichtiger Ausgangspunkt für eine soziologisch orientierte Netzwerkforschung geworden, jedoch

werden Markt, Hierarchie und Netzwerk als voneinander abgrenzbare, eigenständige Typen von sozialer Koordination verstanden, die jeweils über unterschiedliche Koordinationsmittel verfügen. Im Fall des Marktes sind dies Preise, im Fall der Hierarchie bzw. der Organisation formale Regeln, im Fall des Netzwerks Vertrauen (Weyer 2000a: 7).

Der Strukturtyp *Markt* ist in aller Regel gekennzeichnet durch kurzfristige Beziehungen zwischen einem Anbieter und einem Käufer, die Transaktion selbst durch zeitnahen Austausch: Geld gegen Ware. Die Kurzfristigkeit der marktvermittelten Transaktion sorgt dafür, daß die Akteure sich bei jeder Transaktion an der möglichst weitreichenden Durchsetzung ihrer egoistischen Nutzenkalküle orientieren, um entweder einen möglichst hohen monetären Gegenwert für das übertragene Gut zu erhalten oder einen möglichst geringen Preis zu bezahlen.[5] Koordination mittels des Marktes setzt voneinander unabhängige Akteure voraus, die auf einem offenen und sämtlichen interessierten Akteuren zugänglichen Markt freiwillig miteinander in Beziehung treten, um vermittels von Preisen einen Austausch vorzunehmen. Soziale Ordnung ergibt sich dabei aus der Verfolgung individueller egoistischer Nutzenkalküle unter der Rahmenbedingung einer institutionellen Absicherung.[6] Weiteres strukturelles Merkmal dieser Form der Handlungskoordination ist die Offenheit in bezug auf die Marktteilnehmer: Das (idealisierte) Modell setzt voraus, daß sämtliche Personen, die eine Ware bzw. Dienstleistung anbieten oder erwerben wollen, über einen Zugang verfügen und Beziehungen zu Personen mit komplementärem Interesse aufnehmen können.[7]

Der zweite Typus von Handlungskoordination wird meist als *Hierarchie* oder *Organisation* bezeichnet, wobei formale Regeln hier das Koordinationsmittel darstellen, mit denen die Beziehungen zwischen den Akteuren reguliert werden. Hierarchische Formen der Koordination von sozialem Handeln setzen Rangdifferenzen voraus, wobei ranghöhere Akteure durch die Ausgabe von Regeln oder Anweisungen das Handeln der Rangniedrigeren bestimmen oder zumindest deren Verhaltensspielräume beschränken. Das Verhältnis zwischen den Akteuren ist daher als Abhängigkeitsbeziehung zu bezeichnen, was sich insbesondere in Kon-

aufgrund der ökonomischen Verengung auf die Kosten einer Transaktion als den zentralen Faktor für die Wahl eines Modells der Handlungskoordination in die Kritik geraten. Vgl. hierzu ausführlicher Sydow 1992: 145 ff.

5 So zumindest nach Annahme der neo-klassischen Ökonomie.

6 Hierzu zählt nicht nur das Vertragsrecht, sondern auch die Währungs- und Preispolitik. Teilnahme am Markt setzt ein Vertrauen in Regulationsfähigkeit von Konflikten durch das Recht ebenso voraus wie das Vertrauen in die Währungspolitik des Staates.

7 Vgl. auch zusammenfassend Wiesenthal 2000: 51.

fliktfällen zeigt, in denen der Inhaber der höheren Rangposition Machtmittel zur Durchsetzung der Regel anwendet.[8] Diese Form der Handlungskoordination setzt im Vergleich zum Markt einen längeren Zeithorizont voraus: Hierarchiestufen müssen entwickelt und die Akteure diesen zugeordnet werden. Die Frage, was einen Akteur dazu motivieren kann, Regeln oder Anweisungen anderer Akteure zu befolgen und damit einen Eingriff in seinen Handlungsspielraum zuzulassen, läßt sich, da sich hier eine Vielzahl von Anreiz- und Sanktionsmitteln findet, nur in Hinblick auf den Einzelfall beantworten.

Neben diesen beiden aus der Literatur seit den Klassikern soziologischen Denkens bekannten Formen der Koordination[9] bildet der meist dazwischen verortete Typus ‚Netzwerk' eine dritte Form von sozialer Ordnung. Diese Zwischenstellung wird üblicherweise durch die Stärke der Beziehungen zwischen den Akteuren begründet: Während im Fall des Marktes Akteure als autonom und die Beziehungen bei der Hierarchie als abhängig charakterisiert werden, befinden sich die Akteure innerhalb von Netzwerken in interdependenten Verhältnissen zueinander (Mayntz 1992: 25; Weyer 2000a: 9). Diese ‚Zwischenposition' gilt auch in der Dimension Zeit: Hier reicht das Kontinuum von den marktförmigen, kurzfristigen Beziehungen bis hin zur langfristig angelegten Hierarchie. Wiederum dazwischen liegt das Netzwerk, in dem die Akteure ihre Handlungen vor einem mittelfristigen Zeithorizont koordinieren.

Die Eigenständigkeit des Typus Netzwerk wird üblicherweise durch das zum Einsatz kommende Koordinationsmittel Vertrauen begründet. Hierunter soll eine auf Gründen[10] basierende Erwartung (Strasser/Voss-winkel 1997: 218) verstanden werden, daß sich ein (potentieller) Kooperationspartner an explizite oder implizite Vereinbarungen hält. Es handelt sich um einen sozialen Mechanismus, der ein Zeitproblem – Leistungen können in sozialen Beziehungen nur mit zeitlicher Verzögerung ausgetauscht werden – und ein Informationsproblem – es ist unbekannt, ob ein Akteur die von ihm erwarteten Leistungen erbringt – löst (Preisendörfer 1995: 264).[11] Die doppeltkontingente Situation, die daraus

8 Siehe hierzu für den Fall der formalen Organisation Silverman 1970 und Crozier/Friedberg 1979: 46 ff.

9 Für den Markt sei hier an die Arbeiten von Adam Smith, für die formalen Organisationen an die Bürokratie-Studien von Max Weber erinnert.

10 Die Begründetheit der Erwartungen setzt dabei voraus, daß die vertrauende Person sich die Spannweite der Handlungsalternative – eingeschlossen der Möglichkeit des Vertrauensbruchs – vergegenwärtigt und sich erst im Anschluß daran im Wissen um die Riskanz zum Vertrauen entschließt. Beim Fehlen dieses Charakteristikums würde es sich um Zuversicht handeln (vgl. hierzu Luhmann 2001: 149).

11 Vgl. auch Sztomka 1999: 20 ff.

entsteht, daß Zeit- und Informationsprobleme für sämtliche potentielle Kooperationspartner gleichermaßen bestehen, kann nur überwunden werden, indem ein Akteur das Risiko des Handelns übernimmt und darauf vertraut, daß auch der andere die erwartete Handlung ausführt (Krohn 1997a: 240).[12] An die durch den Vertrauensvorschuß stattfindende Asymmetrisierung der Optionen ‚Vertrauen' und ‚Mißtrauen' kann der andere Akteur anschließen, seinerseits Vertrauen gewähren, so daß eine Spirale sich verstärkenden Vertrauens in Gang gesetzt wird. Als Ergebnis kann dann eine dauerhafte und wechselseitige Kooperationsbeziehung entstehen (Preisendörfer 1995: 264). Eine grundlegende Bedingung für den Aufbau von Netzwerken besteht demnach darin, daß die beteiligten Akteure darauf verzichten, ihre Gewinne kurzzeitig zu optimieren, um statt dessen auf der Basis eines aufgeklärten Egoismus (Willke 2001: 138) Rücksichtnahme gegenüber dem Kooperationspartner walten zu lassen und damit das Interesse an Kooperationsgewinnen mittelfristig zu verwirklichen. Die auf Vertrauen basierenden Netzwerke kombinieren dabei Leistungen, die sonst entweder ausschließlich vermittels des Marktes oder ausschließlich durch die Wahl der Koordinationsform Hierarchie zu erhalten wären:

> „Netzwerke stellen Leistungen zur Verfügung, die sonst nur entweder per Markt oder per Hierarchie zu erhalten sind, nämlich die Flexibilität marktförmiger Interaktion und die Verlässlichkeit und Effizienz organisierter Strukturen zugleich. Sie ermöglichen es, Tauschakte durchzuführen, ohne sich auf Unsicherheiten und Risiken marktlicher Transaktionen einlassen zu müssen. Und sie ermöglichen koordiniertes Verhalten, ohne die Rigidität starrer bürokratisch verkrusteter Organisationen in Kauf nehmen zu müssen." (Weyer 2000a: 10)

Durch die relative Autonomie der beteiligten Akteure kommt es typischerweise zu einem Ende der Koordination innerhalb des Netzwerks, sobald die aus dem Netzwerk gezogenen Gewinne ausbleiben, die Kosten der Kooperation die Gewinne übersteigen oder einer der Partner dauerhaft das Vertrauen verletzt.[13]

Nach der knappen Skizzierung dieser in der Netzwerkforschung prominenten Typologie wollen wir uns der Frage zuwenden, inwieweit sie geeignet ist, empirisch beobachtbare Formen der Handlungskoordination zu klassifizieren. Von einer Vielzahl von Autoren wird zumindest Unbehagen oder Kritik geäußert, wenn sie die Enge der Idealtypologie

12 Das allerdings nur zu Lasten des Risikos, das seine Erwartungen durch das opportunistische Verhalten des Anderen enttäuscht werden.

13 Vgl. zum vorangegangenen Abschnitt auch Sydow/Windeler 2000: 12 ff.

beklagen, die eine idealistische Stilisierung darstelle und empirisch kaum Fälle vorfindbar seien, die sich den Typen zuordnen ließen.[14] So finden sich bei Powell eine Vielzahl empirischer Beispiele, in denen Vertrauensbeziehungen zwischen den beteiligten Akteuren zwar vorzufinden sind, die aber – wie bei der von ihm angeführten Film- und Aufnahmeindustrie – schnell in Richtung einer marktförmigen Beziehung kippen können (1996: 231). Auf der Grenze zwischen Netzwerken und Organisationen liegen demgegenüber die sogenannten Franchise-Systeme, bei denen ein fokales Unternehmen aufgrund seiner Ressourcenstärke die Vertragsbedingungen zu seinen Gunsten beeinflussen oder gar diktieren kann. Zwar ist in diesen Fällen eine Langfristigkeit der Kooperation aufgrund von Investitionsentscheidungen der Franchisenehmer und der Laufzeiten der Kooperationsverträge gegeben, ob aber Vertrauen hier das maßgebliche Koordinationsmittel darstellt, darf wohl bezweifelt werden.[15]

Aufgrund der empirischen Vielfalt unterschiedlicher Formen der Organisation von sozialen Beziehungen wollen wir daher die Abgrenzung zwischen den Typen weniger stark betonen, als dies in der Netzwerkforschung bislang häufig geschieht, und damit den ‚Zwischenformen' Platz einzuräumen. Dies gelingt, indem wir zwischen vier Dimensionen unterscheiden und damit rechnen, daß soziale Strukturen – immer gedacht im Rahmen des Netzwerk-Begriffs – breit variieren können:

Erstens gehen wir davon aus, daß die Akteure unterschiedlich stark aneinander gebunden sein können. Diese Dimension wird von uns als Kontinuum verstanden, das bei Netzwerken von starker Abhängigkeit – wie beispielsweise im Fall von Franchise-Systeme – bis zu weitgehender Autonomie – wie beispielsweise im Fall von wissenschaftlichen Fach-Communities – reichen kann.

Zweitens rechnen wir damit, daß der Zeithorizont sozialer Beziehungen in Netzwerken von einer kurzfristigen Handlungskoordination bis hin zu einer lang anhaltenden Beziehung reichen kann. Als Beispiel für den ersten Fall wären Verhandlungssysteme zwischen Tarifparteien, als Beispiel für den zweiten japanische Unternehmenskomplexe zu nennen.

In bezug auf das Koordinationsmittel wollen wir drittens an der Unterscheidung von Preis, formalen Regeln und Vertrauen festhalten, aber

14 Ähnlich argumentiert auch Wiesenthal (2000: 49) mit Blick auf seine Typologie von Markt, Organisation und Gemeinschaft.

15 Vgl. hierzu auch Willke (2001: 149). Dieser entwickelt wegen der Unzulänglichkeiten der Dreier-Typologie Markt, Hierarchie, Netzwerk eine feingliedrigere Typologie mit insgesamt acht unterschiedlichen Formen sozialer Koordination.

gleichzeitig betonen, daß innerhalb von Netzwerken nicht ausschließlich Vertrauen zur Handlungskoordination eingesetzt werden muß, sondern sich verschiedene Koordinationsmittel überlagern können. So findet beispielsweise eine Handlungskoordination in Franchise-Systemen typischerweise durch die in den Verträgen fixierten Regeln statt, während in Zulieferernetzwerken der Preis eine wichtige Rolle spielt.

Hinsichtlich des Modus der Konfliktregulierung wollen wir viertens annehmen, daß zwar die Typen Recht, Macht und Verhandlung prinzipiell unterscheidbar sind, diese jedoch auch in Kombination auftreten können, und es so möglich ist, daß innerhalb eines Netzwerks von einem Modus zu einem anderen übergegangen wird.

Wir verstehen dementsprechend die Typologie von ‚Markt – Netzwerk – Hierarchie‘ als Heuristik, die, sofern die Charakteristika der unterschiedlichen Strukturalternativen, nicht als harte Distinktionsmerkmale verstanden werden, eine flexible Einordnung der Bandbreite empirisch vorliegender Strukturen zuläßt. Die durch die Aufweichung der Typologie verlorene Trennschärfe machen wir durch die genannten vier Untersuchungsdimensionen wieder wett, mit denen bei Bedarf eine nähere Klassifizierung stattfinden kann.

Innovationsnetzwerke – Innovation als rekursiver Prozeß

Im folgenden wollen wir uns dem Konzept der Innovationsnetzwerke zuwenden, das sich für einen besonderen Typus von Gewinn interessiert, der durch eine Kooperation zwischen zwei oder mehrerer Organisationen entsteht: die Generierung von innovationsrelevantem Wissen. Analog zur Transaktionskostenökonomie wird das Zustandekommen *von*, und die Investition *in* Kooperationsbeziehungen durch Gewinne oder strategische Vorteile gegenüber anderen, ausgeschlossenen Akteuren erklärt, so daß das Konzept dem Modell des homo oeconomicus verpflichtet bleibt – auch wenn es sich hier im Vergleich zur Transaktionskostenökonomie um andere Arten von Gewinnen handelt und die Kontingenz der Gewinnrealisierung betont wird.

Der Ansatz der Innovationsnetzwerke wurde in Bielefeld zur Beschreibung der Kooperationsbeziehungen im Bereich des Werkzeugmaschinenbaus entwickelt und bietet eine Lösung der als klassisch zu bezeichnenden Kontroverse zwischen technology-push- und market-pull-Konzepten. Diese Theorietraditionen erklären beide den Zusammenhang von technischem und gesellschaftlichem Wandel, wenn auch mit vertauschten Rollen von Explanandum und Explanans:

Kern von *technology-push*-Konzepten bildet die Annahme, daß durch Forschungs- und Entwicklungsarbeit neue Techniken bzw. technische Verbesserungen hervorgebracht und in den Markt gepreßt werden. Der sicherlich prominenteste Vertreter einer solchen Konzeption ist Joseph Schumpeter, der in seiner Formel der *schöpferischen Zerstörung* die Vorstellung zum Ausdruck bringt, daß neue Konsumgüter, Produktionstechniken, Märkte und Formen der industriellen Produktion neuartige kapitalistische Unternehmungen schaffen, welche die Wirtschaftsstruktur von innen heraus revolutionieren (1950: 137). Die Anwendungen von technischen Innovationen oder die Produktion von innovativen Gütern bilden hier die variierende, erklärende Variable, während die Strukturveränderung im Bereich der Wirtschaft oder in anderen sozialen Bereichen als erklärte, abhängige Variable konzipiert wird.[16] Demgegenüber kritisieren d*emand-/market-pull*-Ansätze die Konzeption von technischem Wandel als exogene und nicht weiter erklärte Variable,[17] nehmen aber letztlich nur einen Austausch von Explanandum und Explanans vor: Dieser Tradition folgend wird der Ursprung von Innovationen im Wandel von Bedürfnissen gesehen, die mit dem zu einem Zeitpunkt zur Verfügung stehenden Wissen befriedigt werden.

„Presumably, what we invent is the joint product of what we want and what we know. That we cannot invent all that we want is certain. That we invent all that we can seems improbable. Roughly speaking, we invent what we can, and in some sense, want badly enough.“ (Schmookler 1966: 12)

Ausgangspunkt von Wandlungsprozessen bildet also die Veränderung von Bedürfnissen, die in Fällen der technischen Machbarkeit und zum Auftreten von neuen Produkten oder Produktionstechniken führt. Bedürfniswandel äußerst sich hier also in Form von Nachfrageveränderungen und führt zu angebotsseitigen Anpassungsreaktionen.[18]

16 Evolutionstheoretisch reformuliert bildet die Variation eine sich wandelnde Technik, die in der Selektionsumgebung Markt mit anderen Techniken konkurriert, bis sich eine Technik stabilisiert. Dies gilt allerdings nur für den Zeitraum, bis weitere technische Innovationen auftreten und eine erneute Konkurrenz stattfindet. Dementsprechend wird die Wirtschaftsform des Kapitalismus hier vor allem unter Prozeßgesichtspunkten analysiert und die dynamischen Wandlungs- und Veränderungsprozesse in der Zeit beschrieben (Schumpeter 1950: 139).

17 Vgl. Schmookler 1966: 4.

18 In evolutionstheoretischer Terminologie ist das Bedürfnis die Variation, während Selektion durch die technische Machbarkeit und ökonomische Marktfähigkeit stattfindet.

Der Ansatz des Innovationsnetzwerks kritisiert nun sowohl die Konzeption von Technik als ‚blinde Variation' innerhalb von technology-push-Theorien (Kowol 1996: 11; Bredeweg/Kowol/Krohn 1994: 192) als auch den Mangel an empirischen Belegen für die These der Zurechenbarkeit der technischen Entwicklung auf einen Bedürfniswandel, den demand-pull-Theorien behaupten (ebd. 190). Wenn sich Technik weder ungerichtet entwickelt noch an ein voll artikuliertes Marktbedürfnis als Orientierungsvorgabe anknüpfen kann, stellt sich die Frage, wie es zu Innovationsprozessen kommt (Krohn 1997a: 236). Die Kritik der Vertreter des Konzepts der Innovationsnetzwerke bezieht sich also auf die von technology-push- und demand-pull-Ansätzen gleichermaßen unterstellten Fähigkeit des Markts, eine Koordination zwischen den Innovationen der Anbieter und den Bedürfnissen von Nachfragern herbeizuführen. Während technology-push-Konzepte die Antwort schuldig bleiben, was einen Inventor zu der hochriskanten Entscheidung veranlassen könnte, ohne die Marktsituation für ein zukünftiges Produkt zu kennen, in dessen Entwicklung große Ressourcenmengen zu investieren, bleiben bei demand-pull-Ansätzen weitgehend unklar, wie die unterstellten Bedürfnisveränderungen auf Seiten der Nachfrager den Anbieter erreichen und dort technische Innovationen induzieren. Der Markt – den die Vertreter beider Theorietraditionen als Ort der Koordination im Auge haben – kann diese Leistung nicht erbringen, da er in mehrerlei Hinsicht Schwächen aufweist.[19] Ursache hierfür ist eine Veränderungsdynamik der Rahmenbedingungen, unter denen die Marktteilnehmer operieren, die sich praktisch in einer Steigerung der Uneindeutigkeit von Marktsignalen und einer Erhöhung des Risikos der Interpretation der Signale niederschlägt. Dafür verantwortlich ist erstens die *Informatisierung* aller Entscheidungsebenen innerhalb von Unternehmen. Diese Entwicklung führt zu einer immensen Erhöhung der Menge an verfügbaren Informationen und als Folge davon zu einer Informationsüberlastung der Entscheider (Kowol 1996: 242). Dies gilt sowohl für Anbieter, die vor der Entscheidung stehen, innovative Produkte zu entwickeln und auf den Markt zu bringen, als auch für Käufer, die das Problem des richtigen Zeitpunkts von Investitionsentscheidungen zu bewältigen haben.

Zweitens lassen sich makrosoziale Wandlungsprozesse beobachten, die als Modernisierung der Modernisierung oder *reflexive Modernisierung*[20] beschrieben werden können. Damit ist ein Veränderungsprozeß gemeint, der sich auf die Wandlungsmuster und Veränderungsdynamik selbst bezieht (Beck/Bonß/Lau 2001: 13). Im Zuge dieses Prozesses

19 So z.B. im Bereich des Werkzeugmaschinenbaus, der durch geringe Losgrößen und Spezialanfertigungen gekennzeichnet ist.

20 Vgl. auch Beck/Giddens/Lash 1996.

werden die Grundmuster des Wandlungsprozesses selbst in Frage gestellt und zentrale Institutionen der Moderne aufgelöst. In der wirtschaftlichen Umwelt von Unternehmen führt dies im Ergebnis zu einer Veränderung des Verhältnisses von Arbeitgebern und Arbeitnehmern, zur Auflösung von Normalarbeitsbiographien, zur Krise der eingespielten korporatistischen Verhandlungssysteme, zur Veränderung der betrieblichen Organisation sowie zum Auftreten von Unternehmenskooperationen, die sich der bekannten Unterscheidung von Markt/Hierarchie nicht fügen und für eine Neubestimmung der Grenzen von Unternehmen sorgen (Döhl et al. 2001: 222 ff.).[21]

Drittens agieren Unternehmen nicht mehr in einer vergleichsweise übersichtlichen, primär nationalstaatlich regulierten Marktumwelt, sondern sind in vielen Branchen in *globale Marktstrukturen* eingebettet. Damit verändert sich die relevante Orientierungsumwelt und zieht die Notwendigkeit nach sich, in einem bislang nicht gekannten Maße Veränderungen von Rahmenbedingungen durch andere staatliche Akteure innerhalb des Weltmarkts sowie das Handeln und die Strategieentscheidungen von räumlich weit entfernten Unternehmen bei eigenen Entscheidungen zu berücksichtigen.[22] Auch diese Entwicklung führt in der Summe zu einer Erhöhung der Komplexität von strategischen Entscheidungen. Die Zumutung der industriellen Moderne gegenüber korporativen Akteuren, Signale der technischen Entwicklung in Handlungsoptionen umzusetzen (Kowol/Krohn 1997: 39), erhöht sich durch Informatisierung, Uneindeutigkeit der Richtung des makrostrukturellen Wandels und Globalisierung. Die Interpretation der Signale aus der Unternehmensumwelt, die Beurteilung zukünftiger Marktentwicklung und die Abschätzung von Innovationschancen unter Berücksichtigung der vorhandenen technischen Kompetenzen wird prekär und kann sowohl auf Seiten der Anbieter als auch auf Seiten der Nachfrager von technischen Innovationen zu einem risikoaversen Verhalten führen (ebd. 40). Nichthandeln und Abwarten bildet dann die häufig gewählte, aber auf Dauer nicht erfolgversprechende Handlungsoption.

---

21 Man denke hier an die Auslagerung von Unternehmensfunktionen durch Outsourcing, die Einrichtung von Profitcentern als eigenständige Wirtschaftseinheiten und an das Entstehen von interorganisationalen Netzwerken.

22 Für die These einer wirtschaftlichen Globalisierung ist es dabei nicht entscheidend, ob wirtschaftliche Verflechtungen tatsächlich zugenommen haben und in einem stärkeren Maße Auslandsinvestitionen getätigt werden oder der Wettbewerbsdruck durch neue Konkurrenten zugenommen hat. Das entscheidende Charakteristikum der Globalisierung liegt vielmehr in der stärkeren Orientierung von Unternehmen an globalen Marktsituationen und in der Veränderung des relevanten Orientierungsrahmens.

Das Konzept der Innovationsnetzwerke hat nun Strukturen zum Gegenstand, die eine Antwort auf die eben beschriebenen Herausforderungen bieten. Es stellt einen theoretischen Rahmen zur Verfügung, mit dem Kooperationen zwischen Akteuren jenseits des Markts analysiert werden können, in denen innovatives Wissen hervorgebracht wird. Es bezieht sich dabei nicht auf vergleichsweise seltene Basisinnovationen – wie den Computer, die Dampfmaschine oder den elektrischen Stromkreis –, sondern auf die weitaus häufigeren sukzessiv-inkrementellen Verbesserungen, deren Charakteristikum ist, daß sie an vorangegangene Entwicklungen und Erfahrungen mit Technik anschließen und daher nicht als technische Revolutionen, sondern als evolutionärer Prozeß im Sinne einer technologischen Konvergenz aufzufassen sind (Asdonk/Bredeweg/Kowol 1994: 69). In ihrem zeitlichen Ablauf ist die Technikentwicklung innerhalb von Innovationsnetzwerken nicht gekennzeichnet durch ein Nacheinander von Invention, Innovation und Implementation, in dem ein Anbieter ein Produkt bis zur Marktreife entwickelt, sondern durch rekursive Rückkopplungen zwischen den beteiligten Akteuren, die es nicht erlauben, den Gesamtprozeß in unterschiedliche Phasen zu zerlegen (Asdonk/Bredeweg/Kowol 1991: 291).

Um die Beziehungen zwischen den unterschiedlichen Akteuren beschreiben zu können, unterscheiden die Vertreter des Konzepts zwischen Herstellern und Anwender, die unterschiedlichen Kontexten (eben Herstellungs- und Anwendungskontexten) entstammen. Hersteller sind Unternehmen, die eine technische Innovation umsetzen, während Anwender Akteure darstellen, die neue Techniken erproben und nutzen. Unterschieden werden weiter zwei grundlegende Typen von Rationalität. Dies ist einerseits die *empirisch-praktische Rationalität*, worunter vortheoretisches, häufig nicht expliziertes Erfahrungswissen betrieblicher Praktiker im Umgang mit technischen Artefakten verstanden wird, und die *theoretisch-wissenschaftliche* Rationalität, die vorrangig durch die Gruppe der Ingenieure in den Forschungs- und Entwicklungsabteilungen verkörpert wird (ebd. 291; 1993: 17).[23] Während das empirisch-prak-

23 Diese Unterscheidung orientiert sich stark an der aristotelischen Differenz von Theorie und Praxis, die dort ein unterschiedliches Ziel der Tätigkeit meint. Während es nach Aristoteles Intention der Suche nach einer Theorie ist, Wissen selbst zu erlangen, ist es Sache der Praxis, einen externen Nutzen herbeizuführen (Bien 1968: 275). Theoretisch-wissenschaftliches Wissen entstammt also einem Kontext, in dem systematisch und methodisch kontrolliert der Erwerb von Wissen betrieben wird, während empirisch-praktisches Wissen durch Tätigkeit und Auseinandersetzung mit Artefakten in einem auf ein äußeres Ziel – wie beispielsweise der Produktion von Gütern – gerichteten Prozeß erfolgt. Trotz dieser unterschiedlichen Handlungsziele bildet aber auch wissenschaftliche Forschungsarbeit eine

tische Wissen immer in einen bestimmten Kontext eingebunden ist – hierfür steht sicherlich paradigmatisch die Verwendung von Werkzeugen, deren Umgang nur bedingt vermittels von Explikation und Anweisung gelehrt werden kann, sondern praktisch eingeübt wird –, handelt es sich bei theoretisch-wissenschaftlichem Wissen um ein dekontextualisiertes Wissen, daß auf dem Prinzip der Trennung, Analyse und Formalisierung beruht. Entscheidend in bezug auf technische Innovationen ist nun, daß diese eben keine bloße Umsetzung des Stands der Technik beziehungsweise der jüngsten ingenieurwissenschaftlichen Kenntnisse darstellen, sondern ihre Entwicklung sowohl auf empirisch-praktischen als auch auf theoretisch-wissenschaftlichen Wissen basiert.[24] Innovationsnetzwerke bilden nun den Ort, an dem diese beiden Formen von Rationalität aufeinandertreffen und in wechselseitigen Lernprozessen neues, innovationsrelevantes Wissen generiert wird:

„Die Spezifikation der Zweckvorstellungen, die in technischen Artefakten eingehen, erfolgt in einem experimentell angelegten Theorie-Praxis-Verhältnis. Sukzessiv werden die zu technisierenden Handlungsabläufe (re-)konstruiert, indem die fertigungstechnischen Probleme aufgespürt und vor dem Horizont technischen Wissens definiert werden. Diese Suchvorgänge können zwar in einem gewissen Umfang theoretisch angeleitet sein, sie stellen aber auf empirisch-praktische Gegebenheiten ab und sind daher durch theoretische oder technologische Vorabdefinitionen nicht allein bestimmbar.“ (Asdonk/Bredeweg/Kowol 1991: 292)

Die Beziehungen zwischen Vertretern unterschiedlicher Rationalitäten werden nun als *Rekursionen* bezeichnet, also als Prozesse, in denen aus Outputs Inputs erzeugt werden, die wiederum zu Outputs verarbeitet werden:

praktische Tätigkeit, bei der Erfahrungswissen anfällt, wie die Laborstudien von Knorr-Cetina (1984) zeigen. Ingenieurstechnische Konstruktionsarbeit, die theoriegeleitet erfolgt, besitzt ebenso eine lokale, historische und personengebundene Einbettung (Heymann/Wegenroth 2001: 118), so daß die Unterscheidung zwischen den beiden Formen von Rationalität eher einen heuristischen Charakter besitzt, was von den Vertretern des Konzepts durchaus bemerkt wird (Asdonk/Bredeweg/Kowol 1991: 290).

24 Vgl. auch Sorge 1985: 50f. Diese gilt häufig auch für die praktische Anwendung von erprobtem Wissen, wie Krohn und Weyer zeigen. Da von vornherein nicht absehbar ist, ob sich ein technisches Handlungssystem in einem neuen Kontext bewährt, das Wissen aber nur durch den Versuch einer Anwendung im Sinne einer experimentellen Implementation zu haben ist, wird die Gesellschaft zum Labor. Die Anwendung von wissenschaftlichem Wissen führt zur Erzeugung von neuem Wissen (dies. 1990: 97 f.).

„Im hier verwendeten Sinne ist ein wechselseitiges Steigerungsverhältnis zwischen Technikentwicklung und -anwendung gemeint. Herstellung und Verwendung von Technik befinden sich in einem gegenseitigen Stimulationsprozeß: mit der Steigerung technischen Wissens wachsen die Erzeugungsmöglichkeiten und umgekehrt führt die Zunahme von neuen Verwendungen, Implementationswissen und Lernprozessen über Rückkopplungsschleifen zu neuem technischen Wissen.“ (Kowol 1998: 75)

Im analysierten Fall des Werkzeugmaschinenbaus arbeiten die Autoren nun zwei Rückkopplungsprozesse zwischen der theoretisch-wissenschaftlichen und der empirisch-praktischen Rationalität heraus. Eine erste Rückkopplung findet sich innerhalb des Herstellungsunternehmens zwischen der Forschungs- und Entwicklungsabteilung und der Werkstattabteilung. Output der FuE-Abteilung bilden technische Zeichnungen und (CAD-)Programme, während die Beiträge der Werkstattabteilung in der Beurteilung der Machbarkeit, Funktionstüchtigkeit und Qualitätssicherung liegen. Eine zweite Rückkopplungsschleife existiert zwischen dem Hersteller- und dem Anwenderunternehmen. Während der Output des Herstellers technische Artefakte bildet, trägt der Anwender im Fall des Werkzeugmaschinenbaus zur Optimierung und Algorithmisierung bei (Asdonk/Bredeweg/Kowol 1991: 293). Zentrales Ergebnis ist dabei, daß Anwender nicht nur die Praktikabilität der gefundenen Lösungen im Sinne einer experimentellen Implementation erproben, sondern auch vor der Übergabe und dem Einfahren der Maschinen mit Veränderungsvorschlägen die Entwicklung und Gestaltung beeinflussen und diese Wissensressourcen dem Hersteller zur Verfügung stellen. Das Innovationsnetzwerk bildet dabei den zentralen Ort, an dem eine Konstruktion von Problemen und Lösungen stattfindet und die Richtung der technischen Entwicklung permanent ausgehandelt wird. Technische Innovationen innerhalb von Innovationsnetzwerken lassen sich somit nicht auf die Tätigkeit eines einzelnen Akteurs zurückführen; die Beiträge zur Entwicklung lassen sich nicht klar zurechnen. Das Ergebnis der Kooperation liegt in der Entwicklung einer *Eigenlösung*. Darunter sind aufeinander abgestimmte Artefakte, neue Organisationsstrukturen und darauf bezogene Kompetenzen auf Seiten der Hersteller und Anwender gezählt (Kowol/Krohn 2000: 141), die nach erfolgreichem Einsatz und bei Aufrechterhaltung der Kooperation Ausgangspunkte für weitere Lern- und Innovationsprozesse bilden können.[25]

25 Innovationsnetzwerke produzieren dementsprechend nicht nur technische Artefakte, sondern bilden Träger von Technisierungsprojekten. Der Terminus betont die mehrdimensionalen Leistungen von Innovationsnetzwer-

Bei der Diskussion der Typologie sozialer Ordnungsstrukturen hatten wir weiter oben Vertrauensbasierung als Merkmal von Netzwerken ausgemacht. Dies gilt in besonderem Maße für Innovationsnetzwerke: Die im Fall des Werkzeugmaschinenbaus an der Kooperation beteiligten Hersteller und Anwender konkurrieren gegen andere, von der Kooperation ausgeschlossene Akteure und sind daher besonders sensibel gegenüber dem Risiko einer Weitergabe von Informationen. Der Schutz von Know-how spielt daher für Hersteller und Anwender eine herausragende Rolle: „Für den Technikhersteller ist es bedeutsam, daß die Testreihen, Daten und Konstruktionspläne zur Produktinnovation nicht zum Konkurrenten gelangen, der Anwender möchte, daß seine Innovationsaktivitäten geschützt werden.“ (Kowol/Krohn 1997: 46) Die Entscheidungsoptionen von Vertrauen und Mißtrauen werden bei der Konstitution von Innovationsnetzwerken durch einen einseitigen Vertrauensvorschuß asymmetrisiert. Einer der Akteure gewährt eine Vorleistung, gibt beispielsweise einen Teil der sensiblen Informationen an den Kooperationspartner weiter und setzt sich damit dem Risiko der Vertrauensenttäuschung aus. Dieser hat nun die Möglichkeit zu einem opportunistischen, kurzfristig nutzenmaximierenden Verhalten, kann aber den Vertrauensvorschuß auch nutzen, um seinerseits Vertrauen zu gewähren. Dadurch kann eine Spirale wechselseitigen Vertrauens in Gang gesetzt werden, die eine Voraussetzung für das Entstehen von rekursiven Rückkopplungsprozessen bildet (Kowol/Krohn 2000: 139).

Abgesichert werden die Vertrauensbeziehungen durch Kontrakte, in denen die zentralen technischen und ökonomischen Parameter festgelegt und zu einer technischen Option verdichtet werden (Kowol/Krohn 1997: 49). Die Kontrakte bieten den Partnern einen nur begrenzten Schutz gegenüber dem Scheitern der Kooperation, verringern zwar die Wahrscheinlichkeit von opportunistischem Verhalten, bilden jedoch kein funktionales Äquivalent für Vertrauen in die Integrität des Kooperationspartners. Die informelle vertrauensbasierte Weitergabe von Informationen ist meist nicht Gegenstand der kontraktuellen Vereinbarungen; weiter müssen die Akteure darauf vertrauen, daß der Partner in der Lage ist, die vertraglich fixierten Leistungen zu erbringen und die Leistungsversprechungen zu erfüllen (dies. 2000: 146).

Zusammenfassend können Innovationsnetzwerke als vertrauensbasierte Kooperationsbeziehungen beschrieben werden, in denen Hersteller und Anwender, die jeweils über unterschiedliche Wissensressourcen verfügen, über einen längeren Zeitraum miteinander kooperieren und

ken in Bereichen von Artefakten, Kompetenzen und Organisationsstrukturen.

durch wechselseitige Lernprozesse innovationsrelevantes Wissen erzeugen. Die Motive für eine Kooperation liegen dabei in informationalen Gewinnen bezüglich der Bedürfnisse der Anwender – also Informationen, an denen ein herstellendes Unternehmen großes Interesse hat – und hinsichtlich der technischen Mach- und Umsetzbarkeit – also Informationen, die bei Investitionsentscheidungen für den Anwender von hoher Bedeutung sind. Diese Informationsgewinne der Kooperation führen zu strategischen Vorteilen gegenüber Konkurrenten auf dem Markt. Dem steht als Kosten das Risiko einer Vertrauensenttäuschung, also der Möglichkeit, daß sich der Kooperationspartner kurzfristig nutzenmaximierend verhält, gegenüber. Dadurch ist das Zustandekommen und die Aufrechterhaltung von Kooperation in Form des Innovationsnetzwerks nur zu erwarten, sofern für sämtliche Kooperationspartner Gewinne in Aussicht stehen – also eine win/win-Situation vorliegt.

Fortgesetzte Kooperation zwischen denselben Akteuren über mehrere Innovationsprojekte hinweg birgt allerdings noch ein weiteres Risiko: das Risiko, auf der Grundlage vergangener Erfolge weitere Innovationsprojekte voranzutreiben und damit die Chance zu verpassen, neue Trends mit zu initiieren oder auftauchende Trends frühzeitig zu adaptieren (dies. 1997: 60). Es besteht also im Fall von langanhaltenden Kooperationen das Risiko, den Optionsraum von weiteren Innovationen aufgrund von positiven Erfahrungen mit dem Kooperationspartner zu eng zu fassen und sich damit in die Gefangenschaft einer technologischen Pfadabhängigkeit zu begeben. Diesem Risiko ist vermutlich nur zu begegnen, indem wechselnde Kooperationen mit unterschiedlichen Partnern gesucht und aufgenommen werden, auch wenn sich hier die Kosten für Vertrauensvorleistungen und die Risiken einer Vertrauensenttäuschung mit der Anzahl der Kooperationspartner multiplizieren.

Teil 1: Die Genese des Entwicklungsnetzwerks

In der Literatur zur freien Softwareentwicklung werden vorrangig Fragen zur Funktionsweise der Entwicklung von Software gestellt, wie zum Beispiel die nach dem Zustandekommen von Beiträgen, dem Einfluß unterschiedlicher Akteure (Unternehmen, Anwender, Entwickler) und nach ökonomischen Erfolgsrezepten, die dann möglicherweise auf den Bereich der proprietären Entwicklung übertragen werden könnten. Zu kurz kommt bei all dem bislang die Entstehung dieser Form der Softwareentwicklung: Sie wird als quasi naturgegeben hingenommen, nicht weiter problematisiert und hinterfragt.

Diese Lücke wollen wir in diesem Teil mit dem folgenden Ausgangspunkt schließen: Wir halten das Entstehen eines sozialen Netzwerks, das Software entwickelt, diese Anwendern ohne direkte Gegenleistung überläßt und es darüber hinausgehend jedem erlaubt, sich bei Interesse an der Entwicklung zu beteiligen, für hochgradig erklärungsbedürftig. Um es anders zu formulieren: Die freie Softwareentwicklung mit ihrer offenen, von jedermann erreichbaren Projektstruktur, die dann von externen Beobachtern als unübersichtlich, ungeordnet oder chaotisch beschrieben wird, kontrastiert auffällig mit dem uns vertrauteren Bild einer Technikentwicklung, die sich hinter verschlossenen Türen in FuE-Abteilungen abspielt. Wie ist das Entstehen also zu erklären? In diesem Kapitel werden wir zeigen, daß die Genese des Netzwerks freier Softwareentwicklung auf zwei Faktoren zurückzuführen ist: (a) auf die rechtliche Innovation freier Softwarelizenzen und (b) auf die organisatorische Innovation der offenen Programmierarbeit.

Die These, die wir entwickeln wollen, lautet, daß diese Innovationen strukturbildende Entwicklungen darstellen, aus denen die grundlegenden und zentralen Eigenschaften des Entwicklungsnetzwerks resultieren. Dabei setzt sich unsere Argumentation zu jedem Faktor aus zwei Bestandteilen zusammen: Zum einen geht es darum zu begründen, weswegen es sich bei freien Softwarelizenzen und offener Programmierarbeit um Innovationen handelt, zum anderen um die Analyse der Konsequenzen, die diese Innovationen für die Entwicklung von Software haben.

Folglich gliedert sich dieser Teil der Arbeit in zwei Kapitel: Im Zentrum des ersten Kapitels steht die rechtliche Innovation der freien Softwarelizenzen. Hierbei geht es nicht um die wiederholte Darstellung der vertraglichen Figur, auf der diese basieren, sondern um die Herausarbeitung deren innovativen Gehalts. Zunächst werden wir die Begründung des Urheberrechts anhand des juristischen Diskurses nachzeichnen und mit der Begründung von freien Softwarelizenzen kontrastieren. Innovativ ist dabei die Perspektive, die mit freien Softwarelizenzen auf den Zusammenhang von Werk, Werkschöpfer und Nutzer verbunden ist. Anschließend werden wir die Konsequenzen und Folgen diskutieren, die freie Software in bezug auf die Entwicklung von Computerprogrammen nach sich zieht. Dabei werden wir sehen, daß die Programme zu einem Kollektivgut werden und Beiträge zur Entwicklung in Form von Gaben geleistet werden können.

Im zweiten Kapitel steht dann die Innovation der offenen Programmierprojekte im Mittelpunkt des Interesses: Die These, die es hier zu entwickeln gilt, lautet, daß diese Innovation einen Bruch in den Vorstellungen über die Organisationsform der Entwicklung von Software markiert, der einschneidender kaum vorstellbar ist. Ebenso wie für die Innovation der freien Softwarelizenzen geht es uns dabei in einem ersten Schritt darum, die Vorstellungen, die bis dato über die Organisation des Softwareentwicklungsprozesses existierten, als Kontrastfolie herauszuarbeiten, um in einem zweiten Schritt die im Zuge dieser Innovation sich abzeichnenden Ideen hinsichtlich der Koordination von Handlungen in Programmierprojekten aufzuzeigen. Dabei wir sich zeigen, daß an die Stelle einer formalen Hierarchie ein Reputationssystem tritt, dessen normative Grundlagen aus einem anderen sozialen Bereich – dem Wissenschaftssystem – überraschend gut bekannt sind.

Die vertragliche Innovation freier Softwarelizenzen

Einen ersten wichtigen Faktor zur Erklärung der Entstehung des Netzwerks freier Softwareentwicklung bilden freie Softwarelizenzen. Wie wir bereits gesehen haben, stellen diese Verträge dar, mit denen die Urheber oder Lizenzgeber den Empfängern weitgehende Nutzungsrechte am Programm einräumen. Sie erlauben einen uneingeschränkten Gebrauch, die Vervielfältigung, die Verbreitung und die Modifikation der durch sie lizensierten Programme, die – zumindest im Fall von ‚copyleft'-Lizenzen – der Einschränkung unterliegen, daß für sämtliche veränderte oder unveränderte Versionen wieder dieselben Lizenzbedingungen gelten. Wie anhand des letztgenannten Modifikationsrechts bereits deutlich wird, berühren sie nicht nur die Distribution und Verwendung, sondern auch die Produktion. Dieser Aspekt steht hier im Mittelpunkt, wenn wir der Frage nachgehen, worin die Innovation freier Softwarelizenzen liegt und welche strukturbildenden Effekte sie für die Entwicklung hat.

Der innovative Gehalt freier Lizenzen erschließt sich in Abgrenzung zu weitverbreiteten Vorstellungen über den Zusammenhang von Werkschöpfer, Werk und Werknutzung. Nach einer knappen Übersicht über die geschichtliche Entwicklung des Urheberrechts werden wir mithilfe einer Rekonstruktion des juristischen Begründungsdiskurses in einem ersten Schritt zeigen, daß mit dem Urheberrecht eine ‚Theorie' über diesen Gegenstandsbereich verbunden ist, in dem das Recht als ein Mittel zur Förderung von schöpferischer Tätigkeit auftaucht, die letztlich im Interesse der Allgemeinheit liegt. Mit der Monopolisierung der Werknutzungsrechte zugunsten des Werkschöpfers wird ein selektiver Anreiz

zur Werkproduktion geschaffen, die im Interesse des Schöpfers, des Verwerters und des Werknutzers liegt.[1]

Nach der Erarbeitung dieser Kontrastfolie können wir in Schritt zwei die innovativen Aspekte von freien Softwarelizenzen bestimmen. Mit der Analyse der Kritik freier Softwareentwickler an der Anwendung des Urheberrechts auf Software werden wir sehen, daß hier eine vom Rechtsbegründungsdiskurs abweichende Gewichtung zwischen den individuellen Belangen und kollektiven Interessen zum Ausdruck gebracht wird. Ein innovatives Moment liegt somit in der neuartigen Perspektive, die auf diesen Zusammenhang entwickelt wird: Die Notwendigkeit der Bereitstellung von selektiven Anreizen für die Förderung von schöpferischer Tätigkeit wird bestritten, Urheberrecht und das Modell der Vergabe von beschränkten Nutzungsrechten vermittels proprietärer Lizenzen als Innovationshemmnis betrachtet.

In einem dritten Schritt kommen wir mit der Frage nach den Konsequenzen freier Softwarelizenzen auf die regulative Dimension zu sprechen. Die Lizenzen stellen eine Vertragsfigur dar, die sich reflexiv auf den durch das Urheberrecht hergestellten Rechtszustand bezieht und die monopolisierten Nutzungsrechte dazu verwendet, die kritisierten Folgen des Urheberrechts zu beseitigen. Damit machen sie freie Software faktisch zu einem Kollektivgut und regulieren die Distribution, sorgen gleichzeitig mit Blick auf die Produktion aber auch dafür, daß Beiträge zur Entwicklung zwangsläufig die Form von Gaben annehmen. Der abschließende Exkurs zu Marcel Mauss ermöglicht es uns dann, die Charakteristika der so geschaffenen Gabenökonomie zu bestimmen.

Die Institution Urheberrecht

Die Geschichte des Urheberrechts ist relativ jung und mit der Entwicklung des Buchhandels eng verbunden. In der Antike und im Mittelalter bestand eine generelle Nachahmungsfreiheit, so daß Werke oder Manuskripte, welche die Sphäre des Verfassers verlassen hatten und durch individuelle Abschriften in Umlauf kamen (Rehbinder 1998: 8), zum Allgemeingut wurden. Diese Situation änderte sich im Zuge der Erfindung des Buchdrucks. Durch die Buchdruckerpresse zur massenhaften Vervielfältigung von Texten entstand ein Ordnungsproblem: Der Buchdruck erforderte hohe finanzielle Investitionen und sorgte für wirtschaftliche

1 Zur Vermeidung von Mißverständnissen sei darauf hingewiesen, daß es hier keineswegs um Rechtskritik geht. Wir zielen mit der Herausarbeitung der kognitiven und regulativen Bestandteile der Institutionen ‚Urheberrecht' und ‚Softwarelizenzen' allein auf die Bestimmung der Differenzen.

Risiken, da eine bestimmte Auflagenhöhe von vornherein festgesetzt werden mußte und damit die Möglichkeit einer Fehleinschätzung der Nachfrage bestand (Wittmann 1991: 26). Ferner bedrohten Nachdrucker die wirtschaftliche Existenz der Erstdrucker, da sich diese die Kosten für Überarbeitung und Suche nach Textvorlagen sparen konnten (vgl. Haberstumpf 1996: 15; Rigamonti 2001: 12 f.). In einem ersten Anlauf wurde das Problem entsprechend der in Europa herrschenden feudalen Gesellschaftsordnung durch die Vergabe von Privilegien gelöst.[2] Später wurden die Gewerbeprivilegien durch werkbezogene Privilegien ersetzt, wobei diese ebenfalls einen Gewerbeschutz darstellten (Bramann/Merzbach/Münch 1995: 123).

Die Privilegierungspraxis kann höchstens als Vorläufer des Urheberrechtes, nicht aber als die Vorwegnahme desselben gelten, da die beiden Regulierungsformen erhebliche Differenzen aufweisen: Erstens unterscheiden sich die Rechtsbegünstigten in sozialer Hinsicht. Waren es im ausgehenden Mittelalter die Vervielfältiger, denen für ein begrenztes Gebiet ein Verwertungsmonopol gewährt wurde, setzte sich mit der Theorie des geistigen Eigentums im Verlauf des 18. und 19. Jahrhunderts eine neue Rechtsauffassung durch, die nun den Schöpfer des Werks zum Begünstigten machte. Zweitens fand durch Privilegien lediglich ein Schutz von Druckerzeugnissen statt, während Abschriften durchaus zulässig waren. Sachlich bezog sich die Regulierung also nur auf eine bestimmte Form, nicht aber auf den Inhalt.

Die Idee eines Urheberrechts fand nur langsam seinen Weg in die Gesetze. Stationen bildeten hier das Preußische Landrecht von 1794, in dem der Schutz des Verlegers vor unbefugter Vervielfältigung mit dem Erwerb des Werks vom Verfasser begründet wurde (Wittmann 1991: 159), und die erstmalige Anerkennung eines geistigen Eigentums des Verfassers im Badischen Landrecht von 1809 (Haberstumpf 1996: 19). Letztgenanntes Recht war aufgrund der kleinstaatlichen Zersplitterung wenig wirksam. Das Bemühen um eine Vereinheitlichung der Rechtsverhältnisse führte 1835 in der Versammlung des Deutschen Bundes zum Erfolg. Der hier getroffene Beschluß bezog sich auf veröffentlichte und unveröffentlichte Werke der Wissenschaften und Künste, sowie auf dramatische Werke und sicherte dem Urheber bzw. seinen Nachkommen ein Nutzungs- und Verwertungsmonopol über das Werk für einen Zeitraum von 10 Jahren nach dem Tod des Schöpfers zu (Grün 1979: 40).

2 Die ersten Privilegien wurden rasch nach der Erfindung der Druckmaschine mit beweglichen Lettern (Johannes v. Gutenberg) vergeben. Diese technische Innovation wird auf die Mitte des 15. Jahrhunderts datiert, das erste überlieferte Privileg wurde in derselben Zeit an Johannes v. Speyer in Venedig vergeben (Grün 1979: 26).

Im Zuge der Verbreitung von Lichtspielen, der Tonkunst und der Photographie zu Beginn des 20. Jahrhunderts, wurde das Urheberrecht auf diesen Gegenstandsbereich ausgedehnt.[3] Die Regulierungen kamen erst nach langer Diskussion über eine angemessene Schutzfrist zustande und sahen eine Monopolisierung der Nutzungsrechte für eine Dauer von 30 Jahren über den Tod des Schöpfers hinaus vor. In den Jahren 1934 und 1965 wurde das Urheberrecht novelliert und die Schutzfrist auf 50 bzw. 70 Jahren post mortem auctoris verlängert (Grün 1979: 44 ff.). Im Jahr 1985 trat ein weiteres Instrument zur Regulierung des Verhältnisses von Werkschöpfer und -nutzer hinzu: Aufgrund massiver Urheberrechtsverletzungen im Zuge der Verbreitung von Vervielfältigungsgeräten wie Fotokopierer und Kassettenrecorder sowie dem Fehlen wirkungsvoller Kontroll- und Sanktionsmittel wurde eine Geräteabgabe eingeführt. Das neu geschaffene Instrument setzt nun nicht das Urheberrecht durch, sondern soll der wirtschaftlichen Absicherung von Werkschöpfern dienen.

Zusammenfassend lassen sich vier Entwicklungstendenzen hervorheben. Erstens läßt sich eine Ausdehnung der Schutzfrist beobachten, die von ursprünglich 10 Jahren post mortem auctoris auf 50 und nun auf 70 Jahre angehoben wurde. Eine zweite Tendenz bildet die räumliche Ausdehnung des Geltungsbereichs des urheberrechtlichen Schutzes, der sich von einem lokalen Schutz durch ein (klein-)staatliches Recht im 19. Jahrhundert hin zu einem globalen Schutz im 21. Jahrhundert entwickelt hat. Drittens erfolgte eine Ausdehnung des Gegenstandsbereichs. Fielen im 19. Jahrhundert lediglich Texte in den Anwendungsbereich, erstreckt er sich heute sowohl auf Film-, Ton- und Bilddokumente als auch auf Texte, Computerprogramme und grafische Benutzeroberflächen.[4] Viertens zeigt die Geschichte des Urheberrechts, daß begleitend zur Entwicklung und Verbreitung von neuen Vervielfältigungstechniken eine fortlaufende Umgestaltung der rechtlichen Regulierungen stattfindet. Dies läßt sich sowohl am Beispiel des im 15. Jahrhunderts erfundenen Buchdrucks, als auch bei der im 20. Jahrhundert sich entwickelnden Fotokopier- und Tonaufnahmetechnik zeigen. Nachdem wir knapp die Entwicklung des Urheberrechts nachgezeichnet haben, wollen wir uns

---

3 In Deutschland 1901 im ‚Gesetz betreffend des Urheberrechts an Werken der Literatur und der Tonkunst' und 1907 im ‚Urheberrecht an Werken der bildenden Künste und der Photographie' (Bramann/Merzbach/Münch 1995: 123).

4 Der Geltungsbereich wird durch § 2 UhRG geregelt. In diesen Bereich fallen Sprachwerke (Schriftwerke Reden, Computerprogramme), Werke der Musik, pantomimische Werke, Tanzkunst, Lichtbildwerke, Filmwerke und Darstellungen wissenschaftlich/technischer Art wie Pläne, Karten, Skizzen (vgl. Fromm/Nordemann 1998; Nordemann 2002: 2).

nun den in der Rechtswissenschaft auftauchenden Begründungen für den Schutz dieser Werke zuwenden und diese rekonstruieren.

Der in der Rechtswissenschaft dominierenden monistischen Theorie[5] zufolge schützt das Urheberrecht die Persönlichkeitsrechte und Vermögensrechte des Werkschöpfers. Nach der persönlichkeitsrechtlichen Komponente wird die Beziehung zwischen Schöpfer und Werk als besonders schutzwürdig angesehen, da das Werk als Ausdruck der Persönlichkeit des Urhebers gilt (Schlicher 1989: 10; Leinemann 1998: 52; Rehbinder 2002: 48). Hieraus entspringen das Recht des Urhebers über die Kontrolle von Ort und Zeitpunkt einer Veröffentlichung sowie der Schutz gegenüber einer Entstellung des Werks. Die vermögensrechtliche Komponente wird uns im folgenden etwas ausführlicher beschäftigen, da hier die für uns im folgenden wichtige Perspektive des Urheberrechts als Instrument zur Steuerung von kulturellen Aktivitäten auftaucht. Ein erster Schritt zur Rekonstruktion der vermögensrechtlichen Seite führt uns zur Person des Werkschöpfers:

„Die Ausgangslage insbesondere für die freiberuflichen Urheber ist denkbar schlecht. Der Urheber trägt das volle Risiko jeden künstlerischen Schaffens […] Da Geisteswerke Ausdrucksformen einer oft eigenwilligen Persönlichkeit sind, kommt es regelmäßig vor, daß der Urheber am Markt vorbei produziert […] Dazu kommt, daß er nie sicher sein kann, ob und wie lange seine schöpferische Kraft fortdauert, ob die nächste Flaute nicht vielleicht doch sein Ende ist." (Schack 2001: 5)

Der Urheber wird uns in diesem Zitat als prekäre Gestalt präsentiert, da seine schöpferische Tätigkeit mehrfach durch Risiken belastet ist – die Fortsetzung seines Erwerbs scheint hiernach alles andere als garantiert. Das Zitat evoziert allerdings die Frage, weswegen gerade Werkschöpfer – seien sie freiberuflich oder nicht – in ihrer Tätigkeit besonders geschützt werden sollten, sehen sich doch andere Berufsgruppen ähnlichen Risiken ausgesetzt: Besteht für Unternehmer das Irrtumsrisiko der Produktion von nicht-vermarktbaren Gütern, laufen körperlich arbeitende Personen Gefahr, die von ihnen zum Einkommenserwerb eingesetzten Kräfte durch Unfall, Krankheit oder Erschöpfung verlieren zu können. Die weitere Begründung führt uns zweitens zu den Charakteristika des vom Urheber geschaffenen Werks, das typischerweise wie folgt beschrieben wird:

5 Während die dualistische Theorie Vermögens- und Persönlichkeitsrechte trennt, geht die monistische Theorie davon aus, daß diese untrennbar sind und sich nur gemeinsam schützen lassen (vgl. Ulmer 1980: 112 f.).

„Geistige Werke haben auch eine ökonomische Besonderheit: Im Gegensatz zu körperlich fixierten Werken ist die Information selbst beliebig oft reproduzierbar. Von ein und demselben Informationsgut lassen sich unendlich viele Kopien erstellen, ohne daß dabei die Information aufgezehrt wird. […] Die Information ist grenzkostenlos mehrnutzbar, d.h. nicht knapp.“ (Leinemann 1998: 59)

Hier interessieren uns zwei Aspekte: Leinemann betont zum einen die durch ihre leichte Reproduzierbarkeit hervorgerufene füllhornartige Eigenschaft, immer in ausreichender Menge vorhanden zu sein. Zum anderen charakterisiert der Autor die Vervielfältigbarkeit als ‚ökonomische‘ Besonderheit, auch wenn es hier durchaus vorstellbar wäre, diese als politisches, religiöses oder wissenschaftliches Charakteristikum zu bezeichnen. Diese Charakterisierung und der Bezug auf die ökonomische Theorie durch ‚Grenzkosten‘ und ‚Knappheit‘ im letzten Satz zeigt an, daß es dem Autor um die Distribution des Werks geht: ‚Knappheit‘ bildet eine im Fall von geistigen Werken als ‚fehlend‘ wahrgenommene Eigenschaft; der Bedarf nach einer rechtlichen Regulation wird also auch aus der Verletzlichkeit des Werks und den Problemen einer marktförmigen Allokation begründet, die zu Schwierigkeiten für den Urheber führen, durch seine Tätigkeit Einkommen zu erwerben.

Neben den Interessen des Urhebers gelten im rechtswissenschaftlichen Begründungsdiskurs auch die anderen Akteure als berücksichtigungswürdig. Dies sind im einzelnen die Interessen der Werksverwerter nach Investitionsschutz, die Interessen der Rezipienten nach leichter Zugänglichkeit und geringen Kosten der Werknutzung sowie die Interessen der Allgemeinheit, zwischen denen das Urheberrecht einen Ausgleich herbeiführen soll (vgl. Rehbinder 2002: 48; Schack 2001: 8; Pahud 2000: 32 f.). Entscheidend ist für uns die Perspektive auf das Allgemeininteresse: „Dieses geht auf dem Gebiet des Urheberrechts dahin, den kulturellen Kommunikationskreislauf aufrechtzuerhalten, was u.a. gebietet, Urhebern und ihren Verwertern den wirtschaftlichen Nutzen aus der Konsumierung von Werken zu sichern.“ (Haberstumpf 2000: 36)[6]

Ungewöhnlich ist, daß das Urheberrecht hier – ebenso wie in der vorangegangenen Begründungen – mit seinen positiven Wirkungen gerechtfertigt wird. Solche konsequentialistischen Argumentationen sind ansonsten in der deutschen Rechtstradition wenig gängig. Das Recht begründet sich aus der positiv bewerteten regulatorischen Wirkung der Aufrechterhaltung des kulturellen Kommunikationskreislaufs, in dessen Existenz die Interessen sämtlicher Gruppen kulminieren: Es ist im Interesse des Werkschöpfers, da sich für ihn Einkommensmöglichkeiten er-

6 Zu analogen Begründungen siehe Pahud 2000: 39; Rehbinder 2002: 48.

schließen, es ist im Sinne der Verwerter, da sich die Möglichkeit ergibt, rentable Investitionen vorzunehmen, und es kommt den Interessen der Rezipienten entgegen, für die Konsummöglichkeiten entstehen.[7]

Die bis hierhin analysierte ‚Theorie' über den Zusammenhang zwischen Werkschöpfer, Werk, Schutzziel und Schutzmittel, die mit dem Urheberrecht verbunden ist, wollen wir thesenartig festhalten: (a) Geistige Werke sind im Unterschied zu körperlichen reproduzierbar, nicht knapp und daher auch nicht marktförmig allokationsfähig. (b) Die Monopolisierung der Verwertungsrechte über einen klar definierten Zeitraum schafft Anreize dafür, schöpferisch tätig zu werden. (c) Anreize führen zu einer Produktion von geistigen Werken; beim Ausbleiben dieser würde die Produktion sich verringern oder zum Erliegen kommen und damit eine Situation entstehen, die weder im Interesse des Werkproduzenten, -verwerter oder -konsumenten liegen kann.

Abschließend wollen wir ein wenig Evidenz für die These herbeischaffen, daß es sich hier nicht um einen juristischen Nischendiskurs handelt, sondern die ‚Theorie' auch Eingang in andere soziale Bereiche gefunden hat, in denen das Urheberrecht eine Rolle spielt. Hierzu sei exemplarisch aus einem offenen Brief von Bill Gates aus dem Jahre 1976 zitiert, der sich an die zu dieser Zeit entstehenden Gruppe der hobbymäßigen Computernutzer wendet und gegen die dort übliche Praxis des Kopierens und Weitergebens der von ihm entwickelten Programmiersprache BASIC folgende Argumente in Anschlag bringt:

> „One thing you do is prevent good software from being written. Who can afford to do professional work for nothing? […] We have written 6800 BASIC, and are writing 8080 APL[8] and 6800 APL, but there is very little incentive to make this software available to hobbists. Most directly, the thing you do is theft. […] Nothing would please me more than being able to hire ten programmers and deluge the hobby market with good software." (Gates 1976, o.S.)

In diesem Zitat finden sich alle weiter oben analysierten Bestandteile wieder: Das verletzliche Werk, der bedrohte Werkschöpfer und – aufgrund der illegalen Vervielfältigung – der zusammenbrechende Kreislauf von Produktion und Distribution des Kulturguts ‚Software'.

---

7 Neben dieser Argumentationslinie wird das Urheberrecht auch durch die eingangs erwähnten, aber bei unserer Rekonstruktion weitgehend unberücksichtigt gebliebenen persönlichkeitsrechtlichen Aspekte gerechtfertigt.

8 Bei APL (A programming language) handelt es sich um eine Programmiersprache, die ursprünglich von Kenneth Iverson an der Harvard University entwickelt wurde. Hier sind Versionen der Sprache für 8080er Intel- und 6800er Motorola-Prozessoren gemeint.

Reflexive Institution: freie Softwarelizenzen

Nachdem wir im vorangegangenen Abschnitt die historische Entwicklung des Urheberrechts sowie die mit ihm verbundene ‚Theorie' analysiert haben, wollen wir uns nun einer anderen Institution zuwenden, die sich reflexiv auf erstgenannte bezieht: freie Softwarelizenzen. Wir wollen zeigen, daß mit freien Softwarelizenzen ebenfalls eine ‚Theorie' verbunden ist, die sich m.E. auf denselben Gegenstandsbereich bezieht, aber zu konträren Schlußfolgerungen kommt. Das Urheberrecht wird hier nicht als Instrument der Förderung schöpferischer Tätigkeit, sondern im Gegenteil als störendes Hindernis wahrgenommen. Dies gilt es im folgenden anhand einiger Texte mit selbstbeschreibendem Charakter der freien Softwareentwickler Richard M. Stallman und Eric S. Raymond herauszuarbeiten.[9] Ein guter Ausgangspunkt zur Rekonstruktion dieser Perspektive bildet die dort vorgebrachte Kritik am Urheberrecht. Thematisiert werden drei Bereiche, in denen negativ bewertete Folgen auftreten: (1) Der Prozeß der Softwareentwicklung selbst, (2) Folgen, die vorrangig Anwender von Software betreffen, sowie (3) Auswirkungen auf die sozialen Beziehungen zwischen den Entwicklern.

Folgen für die Entwicklung von Software: Die Kosten einer Monopolisierung von Werknutzungsrechten für die Entwicklung von Software werden von Stallman wie folgt beschrieben:

> „Software development used to be an evolutionary process, where a person would take an existing program and rewrite parts of it for one new feature, and then another person would rewrite parts to add another feature; in some cases, this continued over a period of twenty years. Meanwhile, parts of the program would be „cannibalized" to form the beginnings of other programs. The existence of owners prevents this kind of evolution, making it necessary to start from scratch when developing a program. It also prevents new practitioners from studying existing programs to learn useful techniques or even how large programs can be structured." (Stallman 1992, o.S.)

Der Autor trifft hier eine Charakterisierung von ‚ursprünglichen' Softwareentwicklungsprozessen. Kennzeichen bildet die evolutionäre Entwicklung, bei der Beiträge unterschiedlicher Personen über längere Zeiträume aufeinander aufbauen und sukzessiv funktionale Erweiterungen implementiert werden. Daneben werden Programme ‚kannibalisiert', wobei damit die Herauslösung und Verwendung von Programmbestand-

9 Wir beschränken uns aus Platzgründen auf diese beiden Autoren. Die Auswahl erfolgte mit Blick auf die breite Rezeption und die große Prominenz dieser Texte im Bereich der freien Softwareentwicklung.

teilen als Ausgangspunkt zur Entwicklung anderer Programme verstanden wird. Diese ursprüngliche oder naturwüchsige Form der Programmentwicklung wird – so die weitere Argumentation – durch das Urheberrecht und der damit verbundenen Monopolisierung von Werknutzungsrechten sowie der Vergabe dieser vermittels proprietärer Lizenzen verhindert, da üblicherweise das Modifikationsrecht nicht mit an den Lizenznehmer übertragen wird. Eine Fortführung dieser Perspektive findet sich bei Raymond, der unter Effizienzgesichtspunkten ausführt:

„Creative brains are valuable, limited resource. They shouldn't be wasted on re-inventing the wheel when there are so many fascinating new problems waiting out there. To behave like a hacker, you have to believe that the thinking time of other hackers is precious.“ (Raymond 2001: 3.2.2)[10]

In beiden Zitaten wird das Urheberrecht, die Monopolisierung von Werknutzungsrechten und der Vergabe dieser durch proprietäre Lizenzen als Hindernis betrachtet, das einer effizienten Softwareentwicklung entgegensteht. Einbußen entstehen dadurch, daß bestimmte Praktiken wie Wiederverwendung und Anpassung durch die Ausübung von monopolisierten Nutzungsrechten unmöglich werden. Die Anwendung des Urheberrechts auf Software führt hiernach zu negativen Externalitäten, da es so einem Softwareautor ermöglicht wird, die von ihm entwickelten Programme anderen Entwicklern als Ressource für ihre Programmierarbeit zu entziehen.

Folgen für die Anwender: Eine zweite Kritik hängt mit der ersten zusammen, stellt allerdings den Anwender mit seinen Bedürfnissen in den Mittelpunkt:

„The ease of modification of software is one of its great advantages over older technology. But most commercially available software isn't available for modification, even after you buy it. It's available for you to take it or leave it, as a black box — that is all.“ (Stallman 1992, o.S.)

Ursache für den Verlust ‚einer der größten Vorteile' von proprietärer Software ist ebenfalls, daß sie üblicherweise ohne Quellcode ausgeliefert wird und Modifikationen durch Softwarelizenzverträge untersagt werden, worunter dann die Nutzbarkeit für die Anwender leidet. Weit-

10 Ein weiteres Effizienzargument findet sich bei Stallman 1992. Durch die Erhebung von Lizenzgebühren würde sich das Programm im Vergleich zur kostenlosen Weitergabe in geringerem Maße verbreiten, so daß sich das Verhältnis des Aufwandes, der für die Produktion eines Programms geleistet werden muß, und dem Nutzen, den ein Programm in der Anwendung erbringt, verschlechtert.

reichender ist aber aus der Perspektive freier Softwareentwickler das Entstehen von Abhängigkeitsbeziehungen zwischen Herstellern und Anwendern:

„When being locked into an unhealthy dependency on your vendor was the only alternative anybody could see, it was hard to evaluate closed source as a risk. But open-source software puts the software consumers back in control of his destiny. It creates a buyer's market for enhancements, service, and support; it allows a mix of options including in-house development or contracting to any one of multiple competing service groups." (Raymond 1999)[11]

Auch hier wird wiederum freie mit proprietärer Software verglichen und dafür argumentiert, erstere sei letzterer überlegen. Nicht nur gewinne der Anwender im Fall der freien Software die bei proprietärer Software verlorengegangene Kontrolle wieder zurück, da er sich von unterschiedlichen Anbietern (anstatt von lediglich einem) mit Support und Service versorgen lassen könne, vielmehr entstehen auch strukturelle Effekte wie die Konkurrenz des Angebots auf dem Markt für mit Software verknüpfter Dienstleistungen, die sich durch proprietäre Software ausschließen läßt.

Folgen für freie Softwareentwickler: Eine dritte negativ thematisierte Konsequenz bezieht sich auf die sozialen Beziehungen zwischen den Entwicklern von Software. Deutlich zum Ausdruck kommt diese im „GNU Manifesto", eine die freie Softwareentwicklung konstituierende Schrift aus dem Jahre 1985:

„Many Programmers are unhappy about the commercialization of system software. It may enable them to make money, but it requires them to feel in conflict with other programmers in general rather than feel as comrades. The fundamental act of friendship among programmers is the sharing of programs; marketing arrangements now typically used essentially forbid programmers to treat other as friends. The purchaser of software must choose between friendship and obeying the law." (Stallman 1985, o.S.)[12]

Der urheberrechtliche Schutz und die Geheimhaltung des Quellcodes konfligiert mit anderen Verhaltenserwartungen, denen Programmierer ausgesetzt sind: Die unter den Softwareentwicklern existierende Norm der Hilfeleistung vermittels Weitergabe und Überlassung von Soft-

11 Dieses Argument findet sich auch bei Stallman 1999: 61, der es als entscheidenden Vorteil des GNU-Betriebssystems ansieht, daß das System die Freiheit des Anwenders respektiert.

12 Vgl. auch Stallman 1999: 54.

ware.[13] Das Verbot dieser Praxis sorgt für die Transformation der sozialen Beziehungen zwischen den Softwareentwicklern und erweist sich so als bedrohlich für die soziale Kohäsion durch die Unterbindung des gruppenkonstituierenden Austauschakts. Die Weitergabe von Programmen hat nicht nur die Funktion, anderen bei der Lösung ihrer Probleme zu helfen, sondern ist darüber hinaus symbolisch stark aufgeladen, wie in der Bezeichnung des Austauschs von Programmen als ‚fundamentalen Akt der Freundschaft' zu Ausdruck kommt. Die Weitergabe bildet die für die Gruppe der Softwareentwickler typische Form, wohlwollende oder freundliche Verbundenheit zu signalisieren, ein Akt, der durch proprietäre Lizenzen wenn nicht verunmöglicht, so doch zumindest verboten wird.

Bereits die bis hierhin analysierten drei Kritikpunkte zeigen, daß die mit dem Urheberrecht und freien Softwarelizenzen institutionalisierten Perspektiven große Differenzen aufweisen. Diese setzen sich nun fort, wenn es um die Frage nach dem Zustandekommen von schöpferischer bzw. innovativer Tätigkeit geht. Eine erste Differenz zeigt sich in der Deutung des Verhältnisses von Werkschöpfer und Werk. Das Urheberrecht geht hier von einer besonderen persönlichen und daher schutzwürdigen Beziehung aus, während die Entwickler einen solchen Zusammenhang zumindest für den Fall von Software bestreiten.

„The emotional argument goes like this: ‚I put my sweat, my heart, my soul into this program. It comes from *me*, it's *mine*!'
This argument does not require serious refutation. The feeling of attachment is one that programmers can cultivate when it suits them; it is not inevitable. Consider, for example, how willingly the same programmers usually sign over all rights to a large corporation for a salary; the emotional attachment mysteriously vanishes. (Stallman 1992, o.S.)[14]

Wie leicht zu sehen ist, paraphrasiert Stallman im ersten Satz die persönlichkeitsrechtliche Begründung des Urheberrechts, die den Anspruch auf die Verfügungsgewalt über das Werk aus der Schöpfungstätigkeit einer Person ableitet. Auch die Annahme, das Werk stelle Ausdruck der Persönlichkeit des Urhebers dar, wird durch die Erwähnung der Seele, die der Schöpfer hingebungsvoll in sein Werk legt, wiedergegeben. In der hieran anschließenden Argumentation bestreitet Stallman die Richtigkeit einer solchen Begründung. Das Werk wird hier nicht als zur Persönlichkeitssphäre des Urhebers zugehörig verstanden und muß demnach nicht

13 Diese normative Orientierung werden wir weiter unten noch eingehender analysieren und als Gabenökonomie bezeichnen.

14 Siehe http://www.gnu.org/philosophy/shouldbefree.html.

vor Veränderungen geschützt werden. Im Gegenteil: Das jedermann gewährte Recht zur Modifikation, wie es sich in freien Softwarelizenzen findet, gestattet es, Abwandlungen, Veränderungen und Ergänzungen vorzunehmen und verweist auf eine Präferenz für die Weiterentwicklung des Programms statt der Bewahrung des Ursprünglichen.[15] Ein zweiter Unterschied bezieht sich auf die Frage der Abwägung zwischen den Belangen des Urhebers und den Interessen an der Werksnutzung anderer Personen.

„Authors often claim a special connection with programs they have written, and go on to assert that, as a result, their desires and interests concerning the program simply outweigh those of anyone else – or even those of the whole rest of the world. […] To those who propose this as an ethical axiom – the author is more important than you – I can only say that I, a notable software author myself, call it bunk." (Stallman 1994, o.S.)

Im ersten Teil des Zitats kommt Stallman erneut auf die der Perspektive des Urheberrechts nach besonderen Beziehung zwischen Schöpfer und Werk zu sprechen, die uns hier nicht weiter interessieren soll. Anschließend macht er deutlich, daß er in der Monopolisierung der Werknutzungsrechte keinen Interessenausgleich sieht, sondern eine einseitige Privilegierung des Urhebers gegenüber anderen. Überraschend an dieser Perspektive auf die Legitimität von Interessen ist dabei, daß er den Akt der Produktion ignoriert und damit den bei der Begründung des Urheberrechts hervorgehobenen Sachverhalt übergeht, daß ein Werk immer die Leistung einer Person oder Personengruppe darstellt. Die Werkschöpfung und der Umstand, daß sich bestimmte Personen daran beteiligt haben und andere nicht, hat dieser Perspektive nach keinen Einfluß auf die Legitimität des Interesses nach einer Werksnutzung und der Frage, wie ein Ausgleich zwischen den Interessen herbeigeführt werden sollte. Das Urheberrecht, das hier eine Asymmetrie der Nutzungsrechte herstellt, indem es den Schöpfungsakt zum Dreh- und Angelpunkt macht (und davon die Legitimität der Nutzungsinteressen ableitet), gilt diesem Gerechtigkeitsverständnis nach als ungerecht.

Die dritte Perspektivendifferenz bezieht sich auf die Beurteilung des Zusammenhangs zwischen dem rechtlichen Schutz von Werken und der Förderung von schöpferischer Tätigkeit. „Fifteen years ago, there were articles on ‚computer addiction': users were ‚onlining' and had hundred-

15 Wäre dem Autor an einem Erhalt des ursprünglichen Programms gelegen, stände ihm der Weg offen, sämtliche Nutzungsformen mit Ausnahme der Modifikation zuzulassen. In diese Richtung tendieren z.T. Lizenzen von Shareware-Programmen.

dollar-a-week habits. It was generally understood that people frequently loved programming enough to break up their marriages.“ (Stallman 1992, o.S.) In dieser Passage wird mit Verweis auf eine die Programmiertätigkeit antreibende Begeisterung die These formuliert, es existierten genügend intrinsische Motive für die Entwicklung von Software,[16] so daß sie nicht weiter durch externe Anreize gefördert werden müsse. Dennoch erweist sich diese Position nicht als blind gegenüber dem Umstand, daß die Entwicklung von Software Ressourcen benötigt.

> „For years, until a fellowship made it unnecessary, I made a living from custom enhancements of the free software I had written. […] Clients paid me so that I would work on the enhancements they wanted, rather than on the features I would otherwise have considered highest priority.“ (Stallman 2002: 48)

Hier tritt deutlich hervor, daß es sich bei freien Softwareentwicklern um Mitglieder einer Profession handelt, die sich aus Leidenschaft und persönlichem Interesse mit einem Gegenstand beschäftigen. Stallman entwickelt die Software, wird aber nicht dafür, sondern nur für die Leistung der Anpassung des Systems an die Bedürfnisse der Anwender bezahlt. Ressourcen müssen demnach also nur bereitstehen, damit Entwickler die Möglichkeit haben, dem intrinsischen Interesse nachzugehen, nicht aber als Anreiz für die Werkschöpfung selbst.[17]

Unsere bisherige Rekonstruktion läßt sich wie folgt zusammenfassen: Bei dem Urheberrecht und den freien Softwarelizenzen handelt es sich um Institutionen, die beide eine ‚Theorie‘ über den Zusammenhang von Werk, Schöpfer und Anreizmuster bereithalten. Entscheidende Differenz ist, daß der Perspektive des Urheberrechts folgend dieses als Instrument der Förderung von Werkproduktion angesehen wird, während in der mit freien Softwarelizenzen verknüpften Perspektive Urheberrecht und proprietäre Lizenzen ausschließlich Innovationshemmnisse darstellen. Gilt das Werk in der Perspektive des Urheberrechts als Ausdruck der Persönlichkeit des Schöpfers, wird dies zumindest im Fall von Software aus der Perspektive freier Softwarelizenzen ebenso bestritten wie die Annahme, die schöpferische Tätigkeit benötige selektive Anreize. Eine weitere Differenz betrifft die Abwägung von Interessen der Werkschöpfer gegenüber denen der -nutzer. Diese äußerst sich darin, daß die

16 Vgl. hierzu auch Raymond 1999: 100; Torvalds/Diamond 2001. Wie wir noch sehen werden, bildet das intrinsische Interesse ein wesentliches Motiv für eine Beteiligung an der freien Softwareentwicklung.

17 Vgl. auch hierzu Raymond 1999. Der gesamte Aufsatz „The Magic Cauldron“ behandelt nichts anderes als unterschiedliche Modelle, mit denen sich mit der Entwicklung von freier Software und damit verknüpften Dienstleistungen Ressourcen mobilisieren lassen (ebd. 156 ff.).

Monopolisierung der Nutzungsrechte der einen ‚Theorie' nach als Mittel des Ausgleichs zwischen den Interessen unterschiedlicher Parteien betrachtet wird, während dies in der anderen ‚Theorie' und der damit verbundenen Gerechtigkeitsvorstellung eine einseitige Privilegierung darstellt.

Konsequenzen für die Entwicklung freier Software

In diesem Abschnitt wollen wir nun der Frage nachgehen, welche Folgen freie Softwarelizenzen haben und welche strukturbildenden Konsequenzen sie nach sich ziehen. Damit ist ihre regulative Dimension angesprochen, die wir in zwei Richtungen entfalten wollen: Zum einen in Richtung der Allokation von Software, zum anderen in Hinblick auf die Produktion. In bezug auf die Allokation werden wir argumentieren, daß freie Softwarelizenzen Computerprogramme faktisch zu einem Kollektivgut machen; hinsichtlich der Produktion wollen wir zeigen, daß Beiträge zur Entwicklung zwangsläufig die Form von Gaben annehmen. Den erstgenannten Punkt werden wir anhand der Literatur zur Kollektivgüterökonomie herausarbeiten, den zweiten in Anschluß an einen Exkurs zu Marcel Mauss' Untersuchungen von Gabenökonomien klären.

Ein Blick auf die Distributionswege von freier Software offenbart eine erste wesentliche Wirkung von freien Softwarelizenzen. Freie Software wird einerseits von den Distributoren über Einzelhandel und Versand gegen die Zahlung einer Gebühr vertrieben, andererseits aber auch auf Internetseiten zum kostenlosen Download bereitgestellt.[18] Der Preis, den die Kunden den Distributoren zahlen, stellt keine Lizenzgebühr dar; bezahlt wird die Leistung der Zusammenstellung der Software zu einer Distribution, die Verbringung der Software auf Speichermedien sowie die Verpackung und Dokumentation. Ebenso wie im Fall des

18 Das gilt sowohl für die einzelnen Softwareprojekte, die auf den Projektseiten aktuelle und ältere Versionen ihrer Programme bereitstellen – wichtige Quellen bilden hier http://sourceforge.net/index.php; http://kde-apps.org; http://www.kde.org/download und http://www.gnu.org/directory/GNU – als auch für komplette Systeme, die sich auf den Seiten der Distributoren finden: http://www.debian.org/distrib/ftplist (für die Debian Distribution); http://www.suse.de/de/private/download/suse_linux (für die SuSE-Distribution); http://www.redhat.com/apps/ download (für die RedHat-Standard-Distribution); http://www.turbolinux .com/support/download (für die TurboLinux-Distributionen). Für Distributionen mit proprietären Erweiterungen müssen z.T. Lizenzgebühren bezahlt werden. Beispiel bildet hier die Mandrake-Distribution: http://www.mandrakesoft.com/products/ range?wslang=de .

Downloads im Internet ist freie Software selbst kostenlos, also ohne die Zahlung einer Lizenzgebühr zu nutzen. Weiter erhält der Empfänger der Software durch die Lizenz das Recht ausgesprochen, die Software zu kopieren und zu vertreiben.[19] Im folgenden Argumentationsschritt wollen wir zeigen, daß freie Softwarelizenzen die hierunter fallenden Computerprogramme zu einem Kollektivgut machen, das durch nichtkonkurrierenden Konsum gekennzeichnet ist.

In der ökonomischen Theorie wird zwischen zwei Typen von Gütern unterschieden: Private Güter und Kollektivgüter.[20] Private Güter oder Individualgüter sind dadurch gekennzeichnet, daß sie einem Besitzer als Eigentum zugerechnet werden, der die Verfügungsgewalt über sie ausübt. Eine Person fungiert als Inhaber der Rechte zum Genuß des Guts, während andere Personen von Nutzung und Konsum ausgeschlossen sind beziehungsweise nur mit der Einwilligung des Besitzers vom Gut profitieren dürfen. In Abgrenzung hierzu werden unter Kollektivgütern diejenigen Güter verstanden, die Mitgliedern einer Gruppe nicht vorenthalten werden können (Olson 1968: 13), weswegen hier auch von Nichtausschließbarkeit gesprochen wird (Hardin 1982: 19; Weber 1980: 65 f.; Ostrom 1999: 38). Einschlägige Beispiele sind Straßenbeleuchtung, innere Sicherheit sowie Radio- und Fernsehprogramme. Das Merkmal der Nichtausschließbarkeit sorgt dafür, daß eine marktförmige Allokation ausfällt, da die Bedingung des Entstehens von Marktpreisen – Zugang zum Genuß wird erst durch einen Kaufakt hergestellt – nicht gegeben ist. Für Nutznießer besteht also im Fall des Kollektivguts keine Veranlassung, ihre Präferenz in Form einer Zahlungsbereitschaft mitzuteilen (Pommerehne 1987: 5) – oder anders gesprochen: es besteht für den Konsumenten kein Anreiz, freiwillig für die Nutzung des Guts zu bezahlen.

Die Nichtausschließbarkeit von kollektiven Gütern führt uns zu einem ersten Problem, das in der ökonomischen Literatur als *Trittbrettfahrer-* oder *free-rider-Problem* bekannt ist. Hiermit ist gemeint, daß trotz

19 Auch dies gilt wiederum nur für Distributionen, die ausschließlich aus freier Software bestehen. Eine bekannte Ausnahme bildete lange Zeit die SuSE-Distribution, deren proprietäre Erweiterung YaST (Yet Another Setup Tool) eine Lizenz besaß, die einen Vertrieb gegen ein Entgelt ohne Einwilligung von SuSE untersagt. Hierdurch verhinderte das Unternehmen, daß Konkurrenz-Distributionen auf Basis der SuSE-Distribution entstanden. Mittlerweile hat SuSE das Programm mit der GPL lizensiert.

20 Vgl. zu dieser Abgrenzung Arnold 1992: 80. Öffentliche Güter stellen eine Unterkategorie von Kollektivgütern dar, die durch die fehlende Entscheidungsmöglichkeit des einzelnen, ob er diese Güter in Anspruch nehmen will oder nicht, und das Fehlen eines rivalisierenden Konsums gekennzeichnet sind.

des gemeinsamen Interesses sämtlicher Mitglieder einer Gruppe an der Existenz und der Möglichkeit der Nutzung eines kollektiven Guts kein individuelles Interesse an der Übernahme der Kosten für die Bereitstellung besteht (Olson 1968: 20).[21] Ein rationaler, egoistischer Akteur[22] zieht es im Fall von Kollektivgütern vor, sich selbst nicht an der Produktion und Bereitstellung der Güter zu beteiligen und die Kosten dafür von anderen tragen zu lassen, um die Vorteile des Guts dann allerdings in Anspruch zu nehmen. Der aus kollektiver Perspektive wünschenswerten anteiligen Kostenübernahme durch jeden Nutzer steht die individuelle Rationalität des Trittbrettfahrens gegenüber, da die zu erbringenden Ressourcen für die Produktion des Guts aus der Perspektive des einsam kalkulierenden Akteurs auch zum Erwerb von individuellen Gütern verwendet werden kann: „Alle Ressourcen, die das Individuum zu einem Kollektivgut beiträgt, bedeutet einen Nutzenentgang im Hinblick auf die Möglichkeit des Konsums von Individualgütern.“ (Weber 1980: 70)

Da sich – zumindest den Modellannahmen der ökonomischen Theorie nach – sämtliche Akteure egoistisch verhalten, wird das Gut nicht produziert, sofern eine Beteiligung auf freiwilliger Basis erfolgt und die Frage von Beteiligung und Nichtbeteiligung von den Akteure selbst entschieden wird. Dementsprechend müssen andere Mechanismen jenseits von Freiwilligkeit gefunden werden, damit das Gut hergestellt und eine Versorgung gewährleistet wird. Beispiele für solche Regelungen bilden die durch Steuern zwangsfinanzierte Produktion der Kollektivgüter durch den Staat, selektive Anreize für Betragszahler und normative Verhaltenserwartungen kombiniert mit sozialer Kontrolle.[23]

Eine weitere Präzisierung des Charakters von freier Software ergibt sich durch die Unterscheidung von konkurrierendem und nicht-konkurrierendem Konsum. Im erstgenannten, häufigeren Fall beeinträchtigt der Genuß des Guts durch eine Person die Nutzungsmöglichkeiten anderer Personen. Beispiele bilden Straßennetz und Parkanlagen. Demgegenüber sind kollektive Güter, bei denen kein konkurrierender Konsum vorliegt, dadurch gekennzeichnet, daß die Inanspruchnahme des Guts durch eine Person die Nutzungsmöglichkeiten anderer Personen nicht

21 Vgl. auch die spieltheoretische Reformulierung des Problems bei Hardin 1982: 25 ff. und Ostrom 1999: 4 ff.

22 Hierbei handelt es sich um eine modelltheoretische Unterstellung innerhalb der ökonomischen Theorie. Weiter unten werden wir die Probleme von Theorien rationalen Wahlverhaltens ausführlicher kritisieren und deren Annahmen verwerfen. An dieser Stelle geht es uns lediglich um eine Charakterisierung der Distribution freier Software und um einen Aufriß der Frage nach der Produktion oder Bereitstellung.

23 Siehe zu den unterschiedlichen Lösungen des Produktionsproblems von Kollektivgütern auch Weber 1980: 75 ff.

beeinträchtigt.[24] Neben dem ersten Problem des Trittbrettfahrens existiert im Fall von Kollektivgütern mit konkurrierendem Konsum ein weiteres, das unter dem Stichwort ‚tragedy of the commons' (Hardin) Prominenz erlangt hat. Dieses Problem wird klassischerweise anhand der Allmende erläutert. Hier nutzt eine Gemeinde kollektiv eine Parzelle Land zum Weiden von Tieren. Sämtliche Mitglieder profitieren von der Verfügbarkeit der Nahrung für die Tiere, sämtliche Mitglieder sind davon gleichermaßen betroffen, sofern nicht genügend Nahrung bereitsteht. Da die Tiere allerdings ein Individualgut darstellen und ausschließlich der Besitzer von ihm profitiert, die Lasten einer Nahrungsknappheit aber alle Nutzer gleichermaßen betrifft, ist es individuell rational, weitere Tiere anzuschaffen und damit einen größeren individuellen Nutzen aus der Verfügbarkeit der Allmende zu ziehen. Da dieses Kalkül – egoistische Orientierung unterstellt – für sämtliche Akteure gilt, besteht im Fall von konkurrierendem Konsum die Tendenz zur Übernutzung. Die Rationalität individueller Nutzenmaximierung steht der kollektiven Rationalität des Erhalts des Kollektivguts entgegen.[25]

Nach der Klärung des Begriffs Kollektivgut sowie der Unterscheidung zwischen konkurrierendem und nicht-konkurrierendem Konsum können wir die Konsequenzen freier Softwarelizenzen für die Distribution und Nutzung der Software näher bestimmen. Ihre schnelle und nahezu aufwandlose Kopierbarkeit in Kombination mit der durch freie Softwarelizenzen gewährten Berechtigung zur Vervielfältigung und zum Vertrieb führen dazu, daß freie Software – einmal in den Umlauf gebracht – zum Kollektivgut wird.[26] Jedermann kann über das Gut verfügen, ein Ausschluß von Personen ist nicht möglich. Es zeichnet sich dabei durch nicht-konkurrierenden Konsum aus. Aufgrund der beliebigen Vervielfältigbarkeit ist es nicht knapp, so daß sich Nutzung wechselseitig nicht beeinträchtigt. Im Gegenteil: Je weiter die Software verbreitet ist, desto leichter ist sie an der Nutzung interessierten Personen zugänglich. Hierdurch kann das weiter oben dargestellte Problem einer Übernutzung des Guts nicht stattfinden – die ‚tragedy of the commons' fällt

24 Man denke hier beispielsweise an innere Sicherheit.

25 Eine Sammlung verschiedener empirischer Beispiel für gemeinsam genutzte, langlebige Allmende-Ressourcen (common-pool-resource) findet sich bei Ostrom 1999: 75 ff.

26 Horns beschreibt freie Software als vertraglich regulierte Allmende (2000: Abs. 40 ff.), also ebenfalls als Kollektivgut. Allerdings ist diese Einordnung irreführend, da es sich bei der Allmende um ein erschöpfliches Gut handelt, das durch konkurrierenden Konsum gekennzeichnet ist, während im Fall von Software durch die leichte Vervielfältigbarkeit keine Knappheit des Guts auftritt. Eine analoge Interpretation von freier Software als Allmende findet sich auch bei Grassmuck 2002: 177 ff.

für freie Software aus.[27] Wie verhält es sich nun mit dem Trittbrettfahrerproblem? Dieses müßte sich, zumindest dem Verständnis der Kollektivgüterökonomie nach, auch im Fall von freier Software aufgrund des Charakteristikums der Nichtausschließbarkeit stellen. Im nächsten Schritt werden wir uns der Frage widmen, warum dies nun gerade nicht der Fall ist, sich also das freerider-Problem – alle nutzen die Ressource, ohne sich an der Bereitstellung zu beteiligen – nicht stellt.

Nicht nur die Distribution und Nutzung, sondern auch die Produktion von freier Software ist in einem erheblichen Maß geprägt von freien Softwarelizenzen: Beiträge zur Entwicklung nehmen aufgrund von freien Softwarelizenzen zwangsläufig die Form von Gaben an, eine andere Form des Beitrags ist ausgeschlossen. Sobald ein Entwickler ein Programm unter einer freien Softwarelizenz lizensiert und es veröffentlicht oder einen Beitrag zu einem freien Softwareentwicklungsprojekt leistet, kann es von anderen Personen unentgeltlich und ohne Gegenleistung genutzt werden. Diese Einordnung des Entwicklungsbeitrags als Gabe gilt nun nicht nur für den Fall eines in seiner Freizeit tätigen, unbezahlten Entwicklers, sondern auch für den Beitrag, der von einem für seine Entwicklungstätigkeit bezahlten Programmierer stammt, wobei sich allerdings die beiden Fälle in bezug auf den Gabengeber unterscheiden. Während im erstgenannten Fall der in seiner Freizeit tätige Programmierer den Geber darstellt, ist dies im zweitgenannten Fall die Organisation, die den Entwickler beschäftigt und für seine Entwicklungstätigkeit bezahlt. Im folgenden wollen wir das Verhältnis von Gabe, Geber und Beschenktem analysieren und zeigen, daß der regulative Aspekt freier Softwarelizenzen in bezug auf die Produktion in der Institutionalisierung einer Gabenökonomie liegt, die besondere Charakteristika aufweist.

Leistungen, die von einer Person erbracht, von einer anderen Personen jedoch nicht aufgrund von rechtskräftigen Verpflichtungen eingefordert werden können, wollen wir im folgenden als Gaben bezeichnen (Clausen 1991: 36). Sie setzen sowohl einen Akt der Übergabe als auch die beteiligten Akteure ego als Geber und alter als Empfänger voraus (Schmied 1996: 11). Leistungsgabe und -empfang werden personal zugerechnet. Bei beiden Beteiligten kann es sich sowohl um Personen als auch um Gruppen handeln, so daß sich zwischen individuellen und kollektiven Gaben auf Seiten des Empfängers und Gebers unterscheiden läßt. Soziale Systeme, in denen die Übergaben von Gaben die Verteilung

27 Zu analogen Schlußfolgerungen kommt Merton im Fall der Wissenschaft, in der wissenschaftliches Wissen als kollektives Gut behandelt und jedermann frei zur Verfügung gestellt wird. Im Zuge der Entwicklung der Wissenschaft vergrößert sich die Menge des zugänglichen Wissens (vgl. Merton 1988: 620).

von Ressourcen regulieren, wollen wir im folgenden als Gabenökonomien bezeichnen. Mit der Funktionsweise solcher Systeme beschäftigt sich Marcel Mauss in seiner anthropologischen Synopse „The Gift" (1954). Mauss hat die grundlegenden Strukturen des Gabentauschs zu einem Modell verdichtet und darin ein Muster zur Herstellung von sozialer Ordnung erkannt.

Hauptuntersuchungsgegenstand von Mauss ist der Potlasch,[28] ein in vielen archaischen Kulturen vorfindbares Ereignis, bei dem der gastgebende Stamm zu Ehren eines besuchenden Häuptlings und dessen Stammesmitgliedern ein Fest ausrichtet. Dabei kommt es zu einem überschwenglichen Konsum von eßbaren und nicht-eßbaren Gütern, der bis zur scheinbar nutzlosen Zerstörung reichen kann:

> „In einigen Fällen geht es nicht einmal um Geben und Zurückgeben, sondern um Zerstörung, nur um nicht den Anschein zu erwecken, als lege man Wert auf eine Rückgabe. Man verbrennt ganze Kisten mit Kerzenfischen (‚candle fish') oder Walfischöl, Häuser und Tausende von Wolldecken; man zerbricht die wertvollsten Kupferplatten oder wirft sie ins Wasser, um einen Rivalen auszustechen, ‚flach zu machen'. (Mauss 1975: 65-67)

Von diesem Phänomen ausgehend interpretiert Mauss den Potlasch nicht nur als ein soziales Ereignis, das der Ernährung, Versorgung und Absicherung dient, sondern als totale soziale Institution, die distinktionsstiftend wirkt und zu einer Rangordnung führt. Durch Verteilung von Gütern, einer besonderen Großzügigkeit und ‚Selbstlosigkeit' im Zuge des Potlasch sowie durch die demonstrative Verachtung von Besitz, die in der Zerstörung von Gütern zum Ausdruck kommt, wird ein Platz in der Hierarchie unter den Häuptlingen und Stämmen erworben. Die Dynamik der wechselseitigen Überbietung und das eskalierende Moment des Potlasch ist allerdings nur zu verstehen, wenn man sich mit Mauss für die institutionalisierten normativen Erwartungen interessiert, die die elementaren Bestandteile der Gabenökonomie bilden. Eine erste Norm bildet die Verpflichtung zur Gabe. Mauss argumentiert hier, Gaben seien nur einer Außenbetrachtung nach freiwillig, müßten faktisch aber aufgrund normativer Verhaltenserwartungen gegeben werden (ebd. 12). Die zweite Norm stellt die Verpflichtung zur Annahme dar (ebd. 27), während die dritte die Verpflichtung zur Erwiderung bildet. Diese drei Normen führen zusammengenommen dazu, daß durch die Ausrichtung eines Potlasch das Recht erworben wird, vom Gast im Gegenzug selbst zu

28 Hierbei handelt es sich um ein Wort aus der indianischen Sprache Chinook, das ursprünglich ‚ernähren', ‚konsumieren' und ‚verbrauchen' bedeutet (vgl. Mauss 1954: 4).

diesem Ereignis eingeladen zu werden: „Die Sanktion der Erwiderungspflicht ist Schuldknechtschaft. […] Es ist dies eine ihrer Natur und Funktion nach mit dem römischen nexum vergleichende Institution. Derjenige, der das Darlehn oder den Potlasch nicht zurückzahlen kann, verliert seinen Rang und sogar den Status eines freien Mannes.“ (ebd. 78)

Das System von Gabe, Annahme und Verpflichtung zur Gegengabe führt nicht nur zu einer Verteilung von Gütern, sondern bildet die Basis für ein Reputationssystem. Reputation wird dadurch erworben, daß den Verpflichtungen zur Gabe, Annahme und Erwiderung nachgekommen wird – und zwar in größerem Maße, als es andere Beteiligte des Reputationssystems tun. Reputationssysteme, die auf Geschenken basieren, bergen somit ein Eskalationspotential, das zu einer stetigen Vergrößerung der Gaben bis hin zur Selbstvernichtung führen kann. Die durch Gaben erworbene Reputation führt nun zu Statusdifferenzen in der Hierarchie der Beteiligten; Verlust von materiellem Besitz wird kompensiert durch einen Zuwachs an Macht (ebd. 16).

Mauss’ Analysen haben ausschließlich den Austausch von Gaben zwischen Kollektiven zum Gegenstand. Als Vertreter von Kollektiven konkurrieren die Häuptlinge in der Öffentlichkeit der beteiligten Clans um Reputation und Rang. Ein Austausch zwischen Kollektiven bildet der Potlasch in sofern, als daß die Mitglieder eines Stammes oder Clans nicht nur unbeteiligtes Publikum bilden, sondern zur Erwirtschaftung der im Potlasch eingesetzten Gaben beitragen. Im Gegenzug profitieren die Mitglieder des Stamms von der erworbenen Reputation, der erworbene Ruf kommt aufgrund eines redistributiven Prinzips allen Mitgliedern zugute. Der Potlasch stellt, nicht zuletzt durch die an das Monströse grenzende Vernichtung von Besitz, ein besonders auffälliges Beispiel für eine Gabenökonomie dar, wird aber vom Autor als Beispiel für ein grundlegendes Muster von sozialem Austausch betrachtet, dessen Elemente auch in anderen Gabenökonomien vorzufinden sind.[29] Die Überlegungen von Mauss sind zu einer wichtigen Grundlage für die Beschäftigung mit wechselseitigen Austauschprozessen avanciert, die durch einseitige Vorleistungen und Verzicht auf eine Festlegung von Gegenleistungen gekennzeichnet sind. Gerade der Ausgangspunkt, die Skepsis gegenüber altruistischen Motiven, hat dabei zur Fokussierung auf verdeckte Funktionen der Gabe geführt und die soziale Dynamik wechselseitiger Verpflichtungen in den Blick gerückt.

29 In einem anderen Kapitel generalisiert Mauss die Befunde auf moderne Gesellschaften und argumentiert, moderne Institutionen wie die französische Sozialgesetzgebung basieren auf der Verpflichtung zur Gabe (ebd. 125).

„Schenken heißt eine Macht erwerben, einen symbolischen Tausch realisieren, Bindungen und Bündnisse initiieren, Rechte und Pflichten attribuieren, subjektive Bedeutungen objektivieren und Alter ego systematisch klassifizieren, heißt strategische Orientierungen in altruistische Motive kleiden, soziale Herausforderungen als Wohltätigkeit stilisieren, beehren, beschämen, hierarchisieren und stratifizieren, solidarisieren, reziproke Anerkennungsformen einfädeln, egalisieren und intimisieren." (Berking 1996: 10)

Deutlicher als Mauss arbeitet Berking zwei Typen von Gabensystemen heraus: den Potlasch, der durch Konkurrenz, Beschämung, Übertrumpfung und letztlich dem Kampf um Hierarchieplätze gekennzeichnet ist, und den undramatischeren Typus des nichteskalierenden Austauschs, der auf Bündnis und Aufrechterhaltung von Beziehungen zielt (ebd. 64). Jedoch ist der Stellenwert dieser Unterscheidung unklar. So läßt sich dagegen einwenden, daß einerseits der Aspekt der Aufrechterhaltung von Beziehungen auch im Fall von Konkurrenz vorfindbar ist und dort gerade die Voraussetzung für die Fortsetzung des Wettbewerbs um Reputation und Rang bildet und daß andererseits auf Aufrechterhaltung von Beziehungen zielende Geschenke auf nichtintendierte Weise in Richtung Konkurrenz ‚kippen' können (vgl. Clausen 1991: 65).[30]

Mit diesem kurzen Exkurs zu Mauss haben wir das grundlegende Verständnis von Gabenökonomien gewonnen, um die Beziehungen und Austauschprozesse zwischen den Entwicklern von freier Software vorläufig einordnen zu können. In einem ersten Schritt sehen wir, daß Beiträge zur Entwicklung von freier Software Gaben bilden, die – im Unterschied zu den von Mauss untersuchten archaischen Kulturen – nicht zwischen Kollektiven ausgetauscht, sondern von einer Person an ein Kollektiv übergeben werden. Die Ursache hierfür liegt zweitens in den freien Softwarelizenzen, deren Bedingungen es erfordern, daß Modifikationen am Quellcode jedermann zugänglich gemacht werden, sie also interessierten Personen ausnahmslos und ohne Gegenleistungen zur Verfügung stehen. Kennzeichnend für diese Form der Gabe ist dabei, daß eine Personalisierung in bezug auf den Geber, nicht aber hinsichtlich des Empfängers stattfindet. Wie der Kreis von Personen, der in den Genuß der Gabe kommt, im einzelnen aussieht, ist aufgrund des Fehlens einer Kontrolle der verschiedenen Vertriebswege von freier Software nicht nachzuvollziehen. Drittens führt die Zurechnung einer Gabe auf eine Person zum Gewinn von Reputation, wie Phänomene der Nennung des

30 Die Autorin unterscheidet daher auch zwischen intrinsischer Motivation (Aufrechterhaltung der sozialen Beziehungen) und extrinsischen Motiven (Verfolgung von Zielen außerhalb der Beziehung, meist der Erwerb von ökonomischen Werten) (Clausen 1991: 38).

Urhebers im Quellcode, bei der Anwendung des Programms[31] und auf der Webseite des Projekts zeigen.

Neben diesen Ähnlichkeiten zu der von Mauss analysierten Gabenökonomie zeigen sich allerdings auch entscheidende Differenzen: Freien Softwareentwicklungsprojekten fehlt es am Moment der Eskalation, das beim Potlasch so auffällig und kennzeichnend ist. Wie läßt sich dieser Befund erklären? Die Ursache für das Ausbleiben von Eskalation liegt darin, daß normative Erwartungen der Gabenökonomie der freien Softwareentwicklung weder wirkungsvoll an Personen adressiert, noch deren Nichteinhaltung sanktioniert werden kann. Praktisch führt dies dazu, daß eine kleine Gruppe von Personen große Beiträge zur Entwicklung leistet, während eine große Gruppe von Akteuren – der weit überwiegende Teil der Anwender – sich nicht an der Entwicklung beteiligt und – aufgrund dessen Anonymität – auch nicht in die Pflicht genommen werden kann. Viertens ist also zu bemerken, daß im Unterschied zur Öffentlichkeit des Potlasch die Einhaltung der Verpflichtungen der Gabenökonomie im Fall der freien Softwareentwicklung nicht extern kontrolliert und Nicht-Einhaltung nicht sanktioniert wird, sondern die Normen offensichtlich eine innere Orientierung der Akteure darstellen. Weiter steht der Eskalation entgegen, daß zwar die Geber von Gaben beobachtet, deren Gaben vom Wert her auch eingeschätzt werden und sich dann auch möglicherweise eine Konkurrenz um die größte Gabe entwickeln kann, sich die Gaben jedoch immer an ein Kollektiv richten. Aufgrund dieser Struktur ist es fünftens nicht möglich, daß es zu der Situation einer persönlichen Schuld durch Ausbleiben einer personenbezogenen Erwiderung kommt.

Welches sind nun die Normen der Gabenökonomie freier Software? Auf den ersten Blick ersichtlich ist, daß diese ohne Verpflichtung zur Annahme der Gabe auskommt. Gaben führen zu einem kollektiven Gut, aus dem sich jedermann bedienen oder dies eben unterlassen kann – eine Norm, die zur Anwendung der Software (und damit zur Annahme der Gabe) zwingt, existiert nicht. Da der Adressat der Gaben immer das Kollektiv sämtlicher potentieller Anwender und sämtlicher Entwickler darstellt, läßt sich nicht mehr in sinnvoller Weise zwischen der Verpflichtung zur Gabe und der Verpflichtung zur Gegengabe unterscheiden: Ob ein Akteur bereits in den Genuß von Gaben gekommen ist und ob er diese nun erwidert oder der Verpflichtung zu einer ersten Gabe nachkommt, hängt letztlich davon ab, ob er freie Software bereits angewendet hat oder nicht. Verpflichtung zur Gabe und Verpflichtung zur Gegengabe fallen also in der Tendenz zusammen – unsere Geschenkökonomie basiert demnach nur auf einer einzigen Norm. Da freie Soft-

31 Häufig durch Nennung der Entwickler auf dem Startbildschirm.

ware immer schon vor einem entsprechenden Beitrag existiert, ziehen wir es vor, diese Norm als ‚Verpflichtung zur Entgegnung der Gabe' zu bezeichnen. Wir fassen diesen Punkt zusammen: Im Unterschied zu Mauss' Potlasch kommt die Gabenökonomie sechstens mit einer einzigen Norm – der Verpflichtung zur Gegengabe – aus.

Die organisatorische Innovation offener Programmierprojekte

Im vorangegangenen Kapitel haben wir die institutionelle Innovation freier Softwarelizenzen analysiert und damit einen Faktor untersucht, der für die Entstehung der kooperativen Form der Programmentwicklung, wie sie im Bereich der freien Software betrieben wird, verantwortlich ist. Bei der Erklärung ihres Entstehens muß jedoch ein zweiter Faktor berücksichtigt werden: offene Programmierprojekte. Das zentrale Argument in bezug auf diese Innovation lautet, daß es sich bei offenen Programmierprojekten um eine institutionelle Neuerung handelt, die mit bis dato existierenden Vorstellungen über den Softwareentwicklungsprozeß bricht und einen neuartigen Orientierungshorizont etabliert. Um diesen Wandlungsprozeß nachzuzeichnen, beginnen wir mit der Darstellung der Geschichte der Softwareentwicklung in den 60er und 70er Jahren, die zur Etablierung der Teildisziplin Software Engineering geführt hat. Von Interesse ist dabei, daß sich hier ein Teilgebiet der Informatik herausbildet, in dem sich das dominante Leitbild zur Organisation der Softwareentwicklung findet. Die Idee ist dort: Die Komplexität der Aufgabe ‚Softwareentwicklung' soll durch weitgehende Formalisierung bewältigt werden.

Die institutionelle Innovation der offenen Programmierprojekte zeigt sich vor diesem Hintergrund erstmalig kurz nach der ersten Veröffentlichung des freien Betriebssystems Linux auf der Newsgroup zum Minix-Betriebssystem. Zwischen Andrew Tanenbaum, Autor von Minix, und Linus Torvalds, Autor von Linux, findet hier eine phasenweise recht scharf geführte Auseinandersetzung statt. Für uns ist diese Diskussion interessant, da hier zwei unterschiedliche Vorstellungen über die Organisation von Softwareentwicklungsprozessen aufeinandertreffen: Während Tanenbaum die aus dem ersten Abschnitt bekannte Position des

Software Engineering vertritt, führt Torvalds – wie rudimentär an dieser Stelle auch immer – die Idee einer selbstorganisierten Entwicklung ins Feld. Dieser Schritt ist entscheidend, macht er doch den Weg frei für eine soziale Organisation des Entwicklungsprozesses jenseits von stark formalisierten Rollen. An diese Leerstelle tritt eine selbstorganisierte Struktur, die weitgehend von Normen geprägt ist. Diese sind vom Wissenschaftssystem her gut bekannt, wie wir in Auseinandersetzung mit Mertons Arbeiten zum wissenschaftlichen Ethos zeigen werden.

Formalisierung des Entwicklungsprozesses: Software Engineering

Von Beginn der Benutzung von Rechenanlagen an bis in die 1970er Jahre hinein wurde Software, die zum Betrieb der Computer notwendig war, überwiegend von Benutzern selbst entworfen und eingesetzt. Computer waren in dieser Frühphase immens teuer und stellten handwerklich gefertigte Einzelexemplare dar (Theißling 1995: 13). Ihr Einsatz beschränkte sich nahezu ausschließlich auf den Bereich der akademischen Forschung und auf militärische Anwendungen (Campbell-Kelly/Aspray 1996; Ceruzzi 2000). Die Rechner wurden in der Maschinensprache programmiert, wobei die Programme häufig nur einmalig eingesetzt und lediglich in seltenen Fällen wiederverwendet wurden. In dieser frühen Phase fielen Programmhersteller und -benutzer in einer Person zusammen; zudem waren computerbezogene Berufe unbekannt. Eindrücklich zeigt dies das Zitat eines Informatikers, der die Probleme einer Anerkennung seines Berufs wie folgt schildert:

> „[I]n 1957, I married, and Dutch marriage rites require you to state your profession and I stated that I was a programmer. But the municipal authorities of the town of Amsterdam did not accept it on the grounds that there was no such profession. And, believe it or not, but under the heading ‚profession‘ my marriage record shows the ridiculous entry ‚theoretical physicist‘.“ (Dijkstra 1979b: 114)

Mitte der 1950er Jahre begannen Unternehmen mit dem Einsatz von Großrechnern, die vor allen Dingen zu Zwecken der Automatisierung von bereits stark formalisierten Arbeitsabläufen eingesetzt wurden. Anwendungen dieser Art wurden unter dem Stichwort *Elektronic Clerk* diskutiert und orientierten sich an dem Bild des Ersatzes von menschlicher Arbeitskraft durch den Computer (Theißling 1995: 16 ff.). Im Unterschied zur Anwendung von Rechnern im Kontext wissenschaftlicher

Forschung wurden Programme zur Lohnabrechnung und Buchhaltung, zur Automatisierung des Scheckverkehrs sowie als Reservierungssystem für Fluggesellschaften nun regelmäßig eingesetzt. Entwickler und -anwender traten auseinander, es entstanden auf die Entwicklung von Software spezialisierte Unternehmen.

Hauptproblem in dieser frühen Phase des Computereinsatzes bildete die möglichst effiziente Nutzung der Hardwareressourcen wie Prozessor, Arbeitsspeicher und Speichermedien, wobei man dies durch kompakt gebaute Programme zu erreichen suchte (Endres 1996: 20). Programmieren galt als eine künstlerische oder trickreiche Tätigkeit (Dijkstra 1979b: 115), bei der es darum ging, die Hardware zu ‚überlisten'. Die Programme, die so entstanden, waren hochgradig individuell und im Ergebnis von den Kompetenzen, dem Programmierstil und den häufig ad hoc umgesetzten Problemlösungen stark abhängig. Das Fehlen einer Standardisierung führte dazu, daß die Programme für andere Personen schwer nachzuvollziehen waren und z.T. vom Programmautor bereits nach kurzer Zeit nicht mehr verstanden wurden. Dieses eigenwillig-individualistische Vorgehen der Entwickler wurde vom Management softwareentwickelnder Unternehmen vor allem als Kontrollproblem wahrgenommen.

„It may be personally satisfying for data processing personnel to be creative and, over a period of time, through their familiarity with certain files or systems, to become indispensable. It is totally *unaccepatable* from a management control point of view to have these valuable and expensive resources operating behind a self-created veil of mystery built on the premise that data processing personnel, projects and activities are different from all other activities in the business." (Gray/Lassiter 1969: 33, zit. nach Theißling 1995: 53)

Ab Mitte der 1960er Jahre und der sogenannten dritten Hardware-Generation[1] beschleunigte sich die Entwicklung im Hardwarebereich. Die Rechenleistung stieg stark an, es wurde nun möglich, größere Programme zu entwickeln und integrierte Systeme aus mehreren Applikationsprogrammen zu produzieren. Obwohl damit die Komplexität – also die Anzahl der Programmbestandteile und die Beziehungen zwischen ihnen – stark anstieg, fand die Entwicklung weiterhin unstrukturiert beziehungsweise nicht durch methodische Prinzipien angeleitet statt (Weber 1992: 26). Während in der Frühzeit des Computers die Hardware das leistungsbegrenzende Nadelöhr darstellte, verschoben sich die Probleme durch die Einführung von Mikroprozessoren: „As long as there were no

1 Kennzeichen dieser ist die Einführung von integrierten Schaltkreisen in die Rechnerarchitektur.

machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem; now that we have gigantic computers, programming has become an equally gigantic problem.“ (Dijkstra 1979b: 116) Die Probleme der Softwareentwicklung, auf die sich Dijkstra hier bezieht, wurden als mehrdimensional wahrgenommen: Hinsichtlich der Software wurde die Fehlerhaftigkeit und die daraus resultierende mangelnde Zuverlässigkeit kritisiert (Dijkstra 1979a: 5). In bezug auf die Entwicklung wurde eine geringe Arbeitsproduktivität bemerkt, welche auf den hohen Anteil an Arbeitszeit zurückgeführt wurde, den die Programmierer mit der Suche und Beseitigung von Fehlern aufwenden mußten (Donaldson 1973: 52).[2] Organisational wurden – wie oben bemerkt – die fehlenden Möglichkeiten zur Steuerung und Leitung der Projekte beklagt, wobei sich dies auf den zeitlichen Horizont der Projekte, die Kosten und das notwendige Personal bezog (Aaron 1970: 70). Mit Blick auf die unterschiedlichen Anwendungsfelder wurde ein Anwendungsstau, also eine Diskrepanz zwischen der Nachfrage und der verfügbaren Software festgestellt (Weber 1992: 40). Daher begann man ab Mitte der 1960er Jahr von einer Softwarekrise zu sprechen, wobei sich diese nicht nur auf das Produkt selbst, sondern explizit auch auf den Entwicklungsprozeß bezog.

„Users complain that the software they received was bad designed and produced, was not delivered on time and when finally delivered, contained serious erros and restrictions and was also inefficient. Unfortunatly, these criticisms are often only too true although, happily, it is unusual for all the criticisms to apply simultaneously.“ (Llewelyn/Wickens 1969: 189)

Als Reaktion auf die Krise wurden Anstrengungen unternommen, den Entwicklungsprozeß stärker zu strukturieren, mit dem Ziel, ihn kontrollieren und lenken zu können. Wichtige Anstöße gingen dabei von zwei NATO-Konferenzen in Garmisch-Partenkirchen (1968) und in Rom (1969) aus, die unter dem Titel Software Engineering abgehalten wurden. Der Titel der Veranstaltungen war in provokativer Absicht gewählt (Naur/Randell 1969: 8), um der Praktik des unsystematischen ‚Herumbastelns‘ am Quellcode das Bild eines ingenieursmäßig organisierten, geplanten und methodisch angeleiteten Entwicklungsprozesses gegen-

2 Das Verhältnis zwischen der Gesamtarbeitszeit und die für Fehlersuche und -beseitigung aufgewendete Zeit war in großen Softwareprojekten dramatisch: „Including all overhead, five to ten debugged instructions are coded per man-day on a large production programming project. The coding time for these instructions cannot exceed more than a few moments of an eight-hour day. What do programmers do with their remaining time? They debug.“ (Baker/Mills: 1973: 58)

überzustellen. Diese beiden Veranstaltungen bildeten dann allerdings den Ausgangspunkt für die Etablierung einer gleichnamigen Subdisziplin innerhalb der Informatik.[3]

Das Programm dieser Krisendisziplin kann dabei als Formalisierung und Strukturierung des Softwareentwicklungsprozesses beschrieben werden. Böhm (1979: 326) zufolge ist Software Engineering „[t]he practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them." Um die Reichweite dieses Vorhabens zu erfassen, werden wir im folgenden einige prominent gewordene Modelle und Werkzeuge vorstellen. Dabei geht es uns darum zu zeigen, daß mit der Etablierung dieser Subdisziplin ein umfassender, neuartiger Orientierungsrahmen geschaffen wird, der nicht nur die Wahrnehmung der während der Softwareentwicklung zu lösenden Aufgabe verändert, sondern auch die Praxis normativ anleitet.[4] Wir beginnen dabei mit der Methode ‚structured programming', die sich auf die sachliche Dimension des Prozesses bezieht.

Ausgangspunkt für die Entwicklung von structured programming bildet ein Papier von Dijkstra mit dem Titel „Programming considered as Human Activity" aus dem Jahr 1965. Darin widmet er sich der Frage, weswegen die Qualität von Software so niedrig sei und macht dafür die begrenzte menschliche Fähigkeit zur intellektuellen Durchdringung von komplexen Strukturen verantwortlich (Dijkstra 1979a: 5). Aufgrund der Beschränktheit der Leistungsfähigkeit des menschlichen Gehirns und der Komplexität der zu lösenden Aufgabe der Softwareentwicklung fordert er dazu auf, die Übersichtlichkeit des Programmquellcodes durch den Verzicht auf sogenannte ‚goto'-Befehle zu verbessern. Dieser Befehl ermöglicht einen Sprung zu einem anderen Teil des Programms und sorgt dafür, daß das Programm kompakt bleibt, beeinträchtigt aber gleichzeitig dessen Les- und die Verstehbarkeit. Der Verzicht auf den Befehl mache das Schreiben von Programmen schwieriger, verringere jedoch den Aufwand für die Beseitigung von Fehlern (ebd. 8). Das Bemühen um die Verbesserung der Struktur des Quellcodes wurde von an-

3 Als Hauptziel dieser Phase der Softwareentwicklung hält Endres fest: „The main goal pursued during this period was to reduce development risk and to improve quality and productivity." (Endres 1996: 22)

4 Neben der Orientierung des Softwareentwicklungsprozesses am Bild des Ingenieurs ist in den 1980er Jahren das Leitbild der Software-Gestaltung getreten, hat aber die ältere Orientierung nicht vollständig verdrängen können (vgl. Schulz-Schaeffer 1996). Zum Verständnis der Innovation von offenen Programmierprojekten sind vor allen Dingen die Konzepte des klassischen Software Engineering wichtig, auf die wir uns in dieser Darstellung beschränken.

deren Informatikern aufgegriffen und durch weitere Vorschläge ergänzt. So erhoffen sich Miller und Lindamood Vereinfachungen der Entwicklung von komplexen Programmen durch ein konsequentes Top-down-Design, der Top-down-Implementation des Systems und der Beschränkung der Größe einzelner Moduleinheiten auf eine Druckseite (Miller/Lindamood 1973: 56). Daneben wurden weitere Werkzeuge wie Hilfsmittel zur Visualisierung und automatisierte Fehlersuchsysteme (Debugger) entwickelt, die Bestandteile von Softwareentwicklungsumgebungen bilden.[5] Es kann also ausgehend von dem Vorschlag von Dijkstra die Etablierung des Interesses an der Entwicklung von Werkzeugen zur Verbesserung der Struktur des Quellcodes nachgezeichnet werden, die später mit anderen Methoden und Werkzeugen des Software Engineering kombiniert werden sollen. Structured programming stellt eine Formalisierung der Sachdimension der Softwareentwicklung dar, mit welcher der Versuch unternommen wird, das Produkt nach bestimmten Kriterien zu standardisieren und ihm seinen stark von den Fähigkeiten des Entwicklers abhängigen Charakter zu nehmen.

Ein weiterer Schritt der Formalisierung – nunmehr in der Zeitdimension – wurde mit der Erarbeitung von sogenannten Vorgehens- oder Life-Cycle-Modellen (Pomberger 1993: 18) genommen. Erstmals wurde ein Vorschlag zur Einteilung des Prozesses in unterschiedliche Phasen mit voneinander abgrenzbaren Problem- und Aufgabenstellungen auf der NATO-Konferenz in Garmisch von Selig präsentiert: Er unterscheidet die Phasen ‚Analysis', ‚Design', ‚Implementation', ‚Installation' und ‚Maintenance' und charakterisiert zentrale Problem- und Aufgabenbereiche, die in der entsprechenden Phase bearbeitet werden (Selig 1969: 13). Unklar bleibt hier allerdings, ob es sich um eine empirische Beschreibung der Softwareentwicklung oder um ein normatives Modell handelt, an dem sich die Praxis orientieren solle.[6] Auch dieses Modell wurde von anderen Informatikern aufgegriffen und durch eine Feingliederung des Prozesses ausgearbeitet.[7] Als entscheidend für eine gelungene Strukturierung der zeitlichen Dimension wird in der Literatur hervorgehoben, daß die Phasen nacheinander durchlaufen und die Aufgaben abschließend gelöst werden sollen, bevor die Probleme einer neuen Pha-

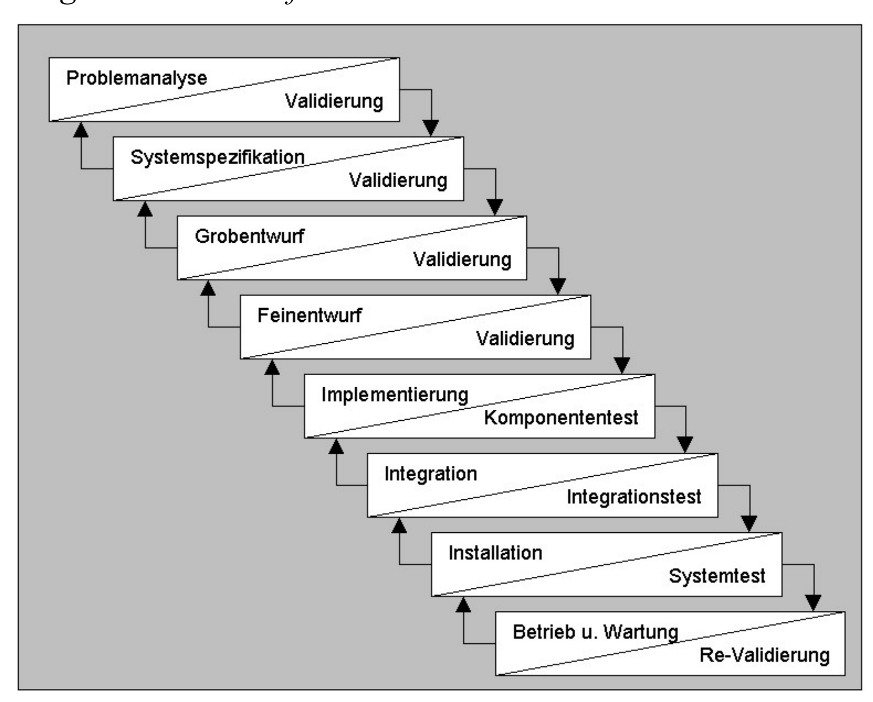
5 Hilfsprogramme für die Softwareentwicklung werden als Computer Aided Software Engineering (CASE)-Tool bezeichnet.

6 Das Modell von Selig ist dem Tagungsband der ersten NATO-Konferenz in Form einer nicht weiter kommentierten Graphik beigefügt. Ob das Modell eher deskriptiv oder normativ gemeint ist, läßt sich nicht beurteilen.

7 Bekanntheit hat hier vor allen Dingen das Modell von Böhm erlangt. Die Phasen seines Spiralmodells decken sich weitgehend mit denen der Wasserfallmodelle (vgl. Böhm 1986: 25).

se bearbeitet werden (z.B. Suhr/Suhr 1993: 98). Auch diese Modelle zielen auf die Erhöhung der Kontrolle über den Softwareentwicklungsprozeß. Das Phasenmodell „ist eine der wesentlichen Grundlagen für einen arbeitsteiligen, einheitlichen, wiederholbaren Prozeßablauf und damit eine Vorbedingung für die Industrialisierung […]. Den Prozeß kann man analysieren, messen und allmählich verbessern." (Chroust 1992: 32) Die folgende Abbildung zeigt das in der Literatur prominenteste Modell, das sich aus insgesamt acht Phasen zusammensetzt, die linear durchlaufen und nur im Problemfall wiederholt werden sollen.

Abbildung 1: Das Wasserfall-Modell



Quelle: Pomberger 1993: 23.

Neben der Formalisierung der Sachdimension und der Modellierung der Zeitdimension wurden auch Bestrebungen unternommen, die Sozialdimension stärker planerisch zu durchdringen. Am bekanntesten ist hier das von Terry Baker und Harlan Mills entwickelte Konzept des ‚Chief Programmer Team'. Dabei handelt es sich ebenfalls um ein normatives Modell der Arbeitsteilung, das die Autoren als „organizational method of increasing programmer productivity" beschreiben (Baker/Mills 1973: 60). Das Konzept fordert, die Entwicklung von Software innerhalb von kleinen Gruppen zu betreiben, die sich aus einem ‚Chief Programmer', einem ‚Backup Programmer', einem ‚Programming Secretary' und weiteren zusätzlichen Programmierern zusammensetzt, die dem ‚Chief Programmer' unterstellt sind (Baker 1979: 67). Im Vergleich zu bis dahin praktizierten Formen der Entwicklung zeichnet sich dieses Modell durch

eine geringe Anzahl von beteiligten Entwicklern und einer klaren Zuordnung von Entscheidungskompetenzen aus, die zu einer erhöhten Produktivität führen sollen.[8] Ergänzt und durch modelltheoretische Überlegungen unterfüttert wurde das Modell von Frederic P. Brooks in seiner einflußreichen Monographie „The Mythical Man Month“: Ausgangspunkt bildet für ihn die Frage, wie sich der Aufwand für die Koordination zur Anzahl der Projektmitglieder verhält. Er stellt fest, daß der Kommunikationsaufwand mit der Anzahl der Mitarbeiter innerhalb eines Projekts ansteigt (Brooks 1995: 18) und entwirft eine mathematische Formel zur Bestimmung des günstigsten Verhältnisses zwischen den Faktoren ‚Teilbarkeit der in einem Projekt bearbeiteten Aufgabe‘ und ‚Anzahl der Projektmitglieder‘. Dies bildet für ihn die Grundlage, um der naiven Vorstellung, mit einer personellen Vergrößerung dem Zeitverzug von Softwareprojekten entgegenwirken zu können, zu entgegnen: „Oversimplifying outtrageously we state Brooks’s Law: Adding manpower to a late software project makes it later.“ (Brooks 1995: 25) Um den Koordinationsaufwand zu begrenzen, greift er das Modell des Chief Programmer Teams auf und propagiert eine hierarchische Projektorganisation.

„[I]n the conventional team the partners are equal, and inevitable differences of judgment must be talked out or compromised. Since the work and ressources are divided, the difference in judgment are confined to overall strategy and interfacing, but they are compounded by differences of interest – e.g., whose spare will be used for a buffer. In the surgical team, there are no differences of interest, and differences of judgement are settled by the surgeon unilaterally. These two differences – lack of division of the problem and the superior-subordinate relationship – make it possible for the surgical team to act uno animo.“ (Brooks 1975: 35)

In der Zusammenschau betrachtet kann die mit der Etablierung von Software Engineering einhergehende Restrukturierung der Softwareentwicklung als Verwissenschaftlichung interpretiert werden. Hierunter verstehen wir in Anlehnung an Weingart einen Prozeß des institutionellen Wandels, der dadurch geprägt ist, daß ältere Handlungsorientierun-

8 Als vieldiskutiertes Beispiel kann hier das „The New York Times Information Bank Project“ gelten. In diesem Projekt wandte IBM erstmalig das Organisationsmodell an und erreichte eine im Vergleich zu anderen Projekten ähnlicher Größenordnung immens hohe Produktivität, die Baker dazu veranlaßte, von der Möglichkeit der Verdoppelung der lines/man month mithilfe des Modells zu sprechen (ders. 1979: 79). Darüber hinaus wurde hervorgehoben, daß der mit dieser Methode produzierte Quellcode nahezu fehlerfrei sei (McCracken 1973: 51).

gen aufgelöst werden (De-institutionalisierung) und vermittels wissenschaftlichem Wissen ein neuartiger Bezugsrahmen geschaffen wird (Re-Institutionalisierung). Im Zuge dieses Prozesses wird der ältere Bezugsrahmen als ineffizient, irrational oder falsch identifiziert (Weingart 1983: 228) und durch einen als angemessener oder dem älteren überlegen geltenden ersetzt. Ein solcher Wandlungsprozeß findet mit der Etablierung von Software Engineering statt, mit der das Produkt „der Kunstfertigkeit einer Gemeinschaft findiger Bastler neubestimmt wird als Resultat eines regelgeleiteten Produktionsprozesses“ (Schulz-Schaeffer 1996: 123). Ausgelöst durch das Fehlschlagen der Martiner 1 Mission,[9] die als ‚Ikone der Softwarekrise‘ firmierte (Campbell-Kelly /Aspray 1996: 200), und dem ungeplanten und als kaum zu steuernd wahrgenommenen Verlauf des IBM OS/360 Projektes[10] veränderte sich die Perspektive auf Softwareentwicklung: Programme wurden spätestens ab diesem Zeitpunkt als wirkmächtig wahrgenommen, konnten sie doch Weltraummissionen scheitern lassen, die Existenz großer Unternehmen gefährden oder Menschenleben aufs Spiel setzen. Gleichzeitig galten sie als komplex, wodurch deren Entwicklung prekär erschien: Innerhalb des Prozesses drohte permanent die Gefahr, Fehler zu machen, die dann unentdeckt in den Systemen verblieben. Vor dem Hintergrund dieser Perspektive auf Software erwies sich das Leitbild des Handwerkers, Bastlers oder Künstlers als unangemessen: Die Bewältigung eines Problems diesen Zuschnitts kann nicht er, sondern nur ein Ingenieur leisten, der über das entsprechende systematische Wissen zur Problemlösung verfügt.

Die obige Inspektion der Toolbox des ‚Software-Ingenieurs‘ offenbart uns nun den Charakter der Verwissenschaftlichung: Zur Anwendung sollen die oben dargestellten Modelle kommen, die sich aus einzelnen (Verhaltens-)Regeln zusammensetzen, deren Einhaltung überprüf- und meßbar ist, und die mittels systematisch-wissenschaftlichen Wissens auf das Erreichen praktischer Resultate zielen. Es handelte sich bei dieser Form der Verwissenschaftlichung also um eine Anwendung

9 Ziel der Mariner 1 Mission (1962) bildete die Erkundung der Venus. Das Raumfahrzeug mußte allerdings bereits vier Minuten nach dem Start zerstört werden, da es aufgrund eines Fehlers im Steuerungsprogramm von seiner Bahn abkam und auf die Erde zu stürzen drohte. Der Fehler wird häufig auch als teuerster Bug der Computergeschichte bezeichnet.

10 Hierbei handelte es sich um die von IBM projektierte Entwicklung eines Betriebssystems für das System 360, dem größten und komplexesten Programm der damaligen Zeit. 1964 geplant, geriet es trotz aller Anstrengungen immer stärker in Verzug und wurde schlußendlich mit mehr als einem Jahr Verspätung ausgeliefert. Dabei beliefen sich die Kosten auf das Fünffache des ursprünglichen Budgets (Campbell-Kelly/Aspray 1996: 199).

von wissenschaftlichem Wissen[11] zur Erreichung des Ziels Qualitätssicherung. Diese Anforderungen stellen nun nicht bloße Erwartungen der akademischen Informatik an die Entwicklungspraxis dar, sondern bilden dort tatsächlich auch eine wichtige Orientierung: So ermitteln beispielsweise Weltz/Ortmann bei einer Untersuchung innerhalb von Softwareunternehmen einen Anteil von 61 Prozent von Entwicklungsprojekten, in denen ein Phasenmodell zur Anwendung kommt (dies 1992: 82),[12] während sich in Fachzeitschriften Erfahrungsberichte über den Einsatz von Methoden, Werkzeugen, Hilfsmitteln und Modellen finden und Lehrbücher der Teildisziplin z.T. aus der Entwicklungspraxis selbst stammen.[13] Die Notwendigkeit einer starken planerischen Durchdringung stellt einen zentralen Orientierungsrahmen für die Softwareentwicklung innerhalb von Unternehmen dar (vgl. Schulz-Schaeffer 1996: 123) und bildet für uns den Hintergrund, vor dem sich die im folgenden zu analysierende Innovation der offenen Programmierprojekte vollzieht.

Die Tanenbaum/Torvalds Diskussion

In diesem Abschnitt wollen wir uns nun der Diskussion zwischen Linus Torvalds, Urheber der ersten Version des Linux-Betriebssystems, und Andrew Tanenbaum, Professor für Betriebssystementwicklung und Entwickler mehrerer Betriebssysteme, zuwenden, da die Diskussion der Gründung einer Linux-Newsgroup vorausgeht, auf der erstmals Software innerhalb eines offenen Programmierprojekts entwickelt wird. In der Diskussion kommt nicht nur eine gegenüber dem Software Engineering alternative Perspektive auf die Organisation der Softwareentwicklung zum Ausdruck, vielmehr tauchen hier bereits die zentralen Bestandteile der Innovation offener Programmierprojekte auf, die dann mit der Gründung der Linux-Newsgroup den organisatorischen Rahmen für die Entwicklung von freier Software abgeben.

Ausgangspunkt für die Entwicklung von Linux bildete ein Projekt des Studenten Linus Torvalds, mit dem er den für sein Studium angeschafften Computer mit einem Intel 386er-Prozessor kennenlernen woll-

11 Daher meinen wir mit Verwissenschaftlichung im Unterschied zu Weingart nicht eine Verlagerung, Ausdehnung oder Verallgemeinerung des Typus des Forschungshandelns auf unterschiedliche gesellschaftliche Bereiche (Weingart 2001: 16), sondern lediglich die Anwendung wissenschaftlichen Wissens zu Zwecken der Softwareentwicklung und eine Orientierung des Vorgehens an einem hohen Maß an Systematik.

12 Bedauerlicherweise wird die Entwicklungspraxis nicht systematisch beforscht, so daß die Datenlage eher schlecht ist.

13 Z.B. Denert 1992.

te. Torvalds anfängliche Beschäftigung zielte dabei nicht auf die Entwicklung eines Betriebssystems, sondern diente der Erkundung der besonderen Architektur und Funktionalität des Chips. Dieses privat betriebene Projekt weitete sich nach und nach aus, die ursprünglichen kleinen Programme wurden von ihm sukzessiv durch andere ergänzt. Ab den Semesterferien im Sommer 1991 wurde das Projekt von Torvalds intensiver betrieben und der Entschluß gefaßt, die einzelnen Programmteile zu einem Betriebssystem mit grundlegender Funktionalität auszubauen.[14] Am 25. August 1991 informierte er mit der folgenden E-Mail auf der Newsgroup *comp.os.minix* andere Personen, die an seinem Betriebssystem Interesse haben könnten, über seine Aktivitäten.

„Hello everybody out there using minix –
I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386 (486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the filesystem (due to practical reasons) among other things).
I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)
Linus (torvalds@kruuna.helsinki.fi)
PS. Yes – it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.“[15]

Der Ort der Veröffentlichung war sorgfältig ausgewählt. Bei der Newsgroup handelt es sich um ein Diskussionsforum von Personen, die am Betriebssystems Minix[16] interessiert sind. Minix stellt ein Unix-ähnliches Betriebssystem dar, das von Andrew S. Tanenbaum zu Lehrzwekken entwickelt wurde. Ziel von Minix war es, den Studenten der Informatik ein Betriebssystem an die Hand zu geben, an dem sie die Grundlagen der Betriebssystemprogrammierung nachvollziehen konnten. Das System kam allerdings nicht nur im Bereich der universitären Lehre zur

14 Unter Betriebssystemen oder Operating Systems (OS) werden in der Informatik Computerprogramme verstanden, welche die Hardware des Computers verwalten. Dazu zählen Rechen- und Steuerwerk, Haupt- oder Arbeitsspeicher, Plattenspeicher und anderer Ein- und Ausgabegeräte.

15 Zitiert a.d. Archiv der Minix-Newsgroup: http://listserv.nodak.edu/scripts/wa.exe?A2=ind9108d&L=minix-l&F=&S=&P=4457.

16 Siehe: http://minix1.hampshire.edu. Grundlegende Informationen zu Minix bietet auch das „Minix Information Sheet“ von Tanenbaum. Siehe: http://www.cs.vu.nl/~ast/minix.html.

Anwendung, sondern verbreitete sich darüber hinaus und wurde u.a. in einer Vielzahl von Unternehmen eingesetzt. Die Newsgroup comp.os. minix diente zur Ankündigung von neuen Versionen und Veränderungen, zur Diskussion über das Betriebssystem, zum Austausch von Erfahrungen sowie zur gegenseitigen Hilfe der Nutzer.

Nach der Veröffentlichung der ersten Linux Version 0.01[17] im September 1991 veränderte sich das Thema der Newsgroup stark: Die Mitglieder begannen über das neue System zu diskutieren, wobei das Thema Minix in den Hintergrund gedrängt wurde. Am 19.01.1992 sendete Andrew Tanenbaum eine Mail mit der Subject-Zeile „Linux is obsolete“ über die Liste,[18] die den Beginn der Tanenbaum/Torvalds Diskussion markiert.[19] Während es im ersten Teil vorrangig um das Für und Wider unterschiedlicher Betriebssystemdesigns geht, steht im zweiten Teil die Frage nach der Organisation von Softwareentwicklungsprojekten im Mittelpunkt. Um die Innovation ‚offener Programmierprojekte’ zu verstehen, lohnt es sich, diese Diskussion etwas ausführlicher zu analysieren.

Ausgangspunkt dieses zweiten Teils bildet eine E-Mail von Andrew Tanenbaum mit der Subject-Zeile „Unhappy Campers“ in der er seiner Beobachtung Ausdruck verleiht, einige Mitglieder der Newsgroup seien mit dem Betriebssystem Minix unzufrieden. Dies bildet für ihn den Ausgangspunkt für die Frage nach den Gründen dafür, wobei er in seiner E-Mail als Ursachen die Lizenzgebühr sowie die Bedingungen der Minix-Lizenz[20] diskutiert und zum folgenden Schuß kommt:

„I think the real issue is something else. I’ve been repeatedly offered virtual memory, paging, symbolic links, window systems, and all manner of features.

17 Siehe zur Entwicklung http://www.kernel.org.

18 Auslöser hierfür bildete nicht zuletzt die oben geschilderte thematische Verschiebung des Gegenstands der Newsgroup zugunsten von Linux.

19 Dies wird jedenfalls von den Akteuren des Feldes so gesehen, da hier die Dokumentation der Diskussion auf einer Vielzahl von Webseiten (und in DiBona/Ockman/Stone 1999: 222 ff.) einsetzt. Vollständig scheint allerdings nur das hier verwendete Archiv der Minix-Newsgroup zu sein: http://listserv.nodak.edu/scripts/wa.exe?A0=minix-l.

20 Die ursprüngliche Lizenz räumte den Empfängern die Nutzung und Modifikation des Programms ein, während eine Weitergabe nur zu Forschungszwecken und für die Ausbildung erlaubt war. Sämtliche andere Formen der Weitergabe von modifizierten Versionen waren hingegen verboten bzw. bedurften der Einwilligung des Halters der Copyrights (Prentice Hall) (vgl. http://www.pdos.lcs.mit.edu/ld/MINIX-README.html). Die Nutzungsbedingungen änderten sich im Jahr 2000, als Minix unter die BSD-Lizenz gestellt und damit zu freier Software wurde (vgl. http://minix1 .bio.umass.edu/LICENSE.html).

I have usually declined because I am still trying to keep the system simple enough for students to understand. You can put all this stuff in your version, but I won't put it in mine. I think it is this point which irks the people who say ‚Minix is not free,' not the $60." (Tanenbaum, A.S.: 3 Feb. 92 22:46:40)

Entscheidend für den uns interessierenden Zusammenhang ist hier, daß Tanenbaum die Diskrepanzen zwischen seinen eigenen Zielsetzungen und denen eines Teils der Newsgroup verantwortlich für die Unzufriedenheit macht. Während es ihm selbst darum geht, über ein übersichtliches Lehrsystem zu verfügen – und daher Modifikationen ablehnend gegenübersteht –, sind andere Mitglieder der Newsgroup an funktionalen Erweiterungen und Ergänzungen interessiert. Die konservative bzw. restriktive Haltung Tanenbaums konfligiert also mit der Innovationsfreude anderer Personen. Warum ist nun aber diese ablehnende Haltung für den am Programm interessierten Personenkreis überhaupt von Bedeutung und von einer solchen Relevanz, daß es zu einer Diskussion kommt? Der Schlüssel zum Verständnis liegt in der besonderen Rolle von Tanenbaum, die er in bezug auf das Betriebssystem einnimmt. Er ist derjenige, der über das alleinige Recht zur Veröffentlichung verfügt, so daß das System nur diejenigen Bestandteile beinhaltet, die er in die Version aufnimmt. Ergänzungen oder Weiterentwicklungen, die von ihm nicht aufgenommen werden, müssen von interessierten Nutzern separat nachinstalliert werden – eine inkrementelle Entwicklung des Systems ist unter diesen Bedingungen nicht möglich. Ebenfalls durch die Lizenzbedingungen verhindert wird die Veröffentlichung einer alternativen Version, die eine von Tanenbaum unabhängige Weiterentwicklung ermöglichen würde. In der Summe führt dies dazu, daß interessierte Programmierer von der Entwicklung ausgeschlossen sind.

In einem zweiten Teil der E-Mail kommt der Autor auf Linux zu sprechen und diskutiert die Möglichkeit einer ‚freien' Programmentwicklung. Dabei nimmt er die im Software Engineering institutionalisierte Perspektive auf Softwareentwicklung ein. Konkreter: Tanenbaum schließt eine erfolgreiche Entwicklung eines Systems bei Verzicht auf klare Verteilung von Entscheidungskompetenzen aus:

„An interesting question is whether Linus is willing to let Linux become ‚free' of his control. May people modify it (run it?) and sell it? […] Suppose Fred van Kempen returns from the dead and wants to take over, creating Fred's Linux and Linus' Linux, both useful but different. Is that ok? The test comes when a sizable group of people want to evolve Linux in a way Linus does not want. Until that actually happens the point is moot, however." (Tanenbaum, A.S.: 3 Feb. 92 22:46:40)

Die an dieser Stelle nur angedeutete und hier noch unausgesprochene Gefahr einer liberalen Handhabung des Modifikationsrechts und einem Verzicht auf eine hierarchische Projektsteuerung liegt Tanenbaum zufolge in einer Verzweigung des Programmquellcodes und dem Entstehen von zunehmenden Inkompatibilitäten zwischen den unterschiedlichen Versionen des Programms.[21] Dementsprechend begründet er die Monopolisierung der Kontrolle über die Weiterentwicklung von Minix mit der Vermeidung der Zerfaserung des Programms in inkompatible Versionen. Dabei kommt im letzten Satz der Zweifel zum Ausdruck, daß die Freiheit der Entwicklung im Fall von Linux größer sei als bei Minix: Freiheit zeige sich erst im Konflikt zwischen unterschiedlichen Entwicklungszielen und der Möglichkeit von Abweichlern, ihre Ziele zu verfolgen. Zwei Tage später führt Tanenbaum seine Überlegungen zur Programmentwicklung unter der Bedingung eines Verzichts auf eine starke Hierarchie aus, und konkretisiert die darin liegende Problematik:

„The problem is co-ordinating things. Projects like Gnu, Minix, or Linux only hold together if one person is in charge. During the 1970s, when structured programming was introduced, Harlan Mills pointed out that the programming team should be organized like a surgical team – one surgeon and his or her assistants, not like a hog butchering team – give everybody an axe and let them chop away. Anyone who says you can have a lot of widely dispersed people hack away on a complicated piece of code and avoid total anarchy has never managed a software project.“ (Tanenbaum, A.S.: 5 Feb. 92 23:23:26)

Tanenbaum nimmt hier explizit bezug auf die uns von der Analyse des Software Engineering her bekannte Perspektive, Arbeitsteilung und kooperative Programmentwicklung seien im Fall von größeren Projekten nur unter der Bedingung einer klaren Zuständigkeit und eindeutigen Zuteilung von Entscheidungskompetenz möglich. Sofern diese Voraussetzungen nicht gegeben seien, würden Koordinationsprobleme zwischen den von unterschiedlichen Personen verfolgten Zielen auftreten, die letztlich zur Aufspaltung des Projekts führen. Aus der von Tanenbaum eingenommen Perspektive des Software Engineering ist der Verzicht auf ‚Kontrolle' über die Programmentwicklung eine Unmöglichkeit.

Die Entgegnung von Linus Torvalds ist vor dem Hintergrund dieser im Brustton der Überzeugung vertretenen Position überraschend: Mit ähnlicher Vehemenz weist er die von Tanenbaum lediglich angedeutete

21 Diese Erfahrungen waren vermutlich einem Großteil der Newsgoup-Mitglieder von Unix her vertraut, das aufgrund der Entscheidungen verschiedener Hersteller in unterschiedliche Versionen zerfiel, die zunehmend inkompatibel wurden (siehe McKusick 1999).

Spekulation, er könne heimlich oder verdeckt die Entwicklung des Programms steuern, zurück.

„Just so that nobody takes his guess for the full thruth, here's my standing on ‚keeping control', in 2 words (three?): I won't.
The only control I've effectively been keeping on linux is that I know it better than anybody else, and I've made my changes available to ftp-sites etc. Those have become effectively official releases, and I don't expect this to change for some time: not because I feel I have some moral right to it, but because I haven't heard too many complaints, and it will be a couple of months before I expect to find people who have the same ‚feel' for what happens in the kernel." (Torvalds, L.B.: 6 Feb. 92 10:33:31)

Dieser lapidar dahingeschriebene Satz, Torvalds plane es nicht, die Kontrolle über das Projekt ausüben zu wollen, markiert einen tiefen Bruch mit dem Software Engineering. Die dort als Selbstverständlichkeit geltenden formalisierten Rollen als Voraussetzung erfolgreicher Programmentwicklung werden hier für entbehrlich gehalten. Was tritt nun aber an diese Leerstelle? Hier finden sich zwei Hinweise: Mit der Verfügungsgewalt über die FTP-Site[22] und seiner ‚Intuition' bezüglich der Vorgänge im Kernel spricht er zwei Faktoren an, die ihm ein gewisses Maß an Kontrolle über das Projekt verschaffen. Die FTP-Site und die darauf veröffentlichte Version sind für eine kooperative Programmentwicklung von zentraler Bedeutung, da damit ein bestimmter Stand der Entwicklung definiert wird, der gleichzeitig als Ausgangspunkt für Weiterentwicklungen gilt. Torvalds – als die Person, die über das Schreibrecht auf dieser Seite verfügt – nimmt damit eine strategische Position ein. Er hat die Möglichkeit, bestimmte Beiträge nicht zu akzeptieren und Weiterentwicklungen in eine vom ihm unerwünschte Richtungen zu bremsen.[23] Diese Rolle eines Gatekeepers, der ‚offizielle' Versionen des Programms veröffentlicht, trägt nun allerdings stark informelle Züge. Aufgrund der Lizenzbedingungen steht es anderen Personen frei, alternative Versionen auf anderen FTP-Sites zu veröffentlichen und Dritten

22 FTP meint file-transfer protocol. Es handelt sich dabei um einen Dienst, der auf TCP/IP (Transfer Control Protocol/Internet Protocol) aufsetzt und es ermöglicht, eine beliebige, in einem FTP-Verzeichnis befindliche Datei über ein Rechnernetzwerk auf einen anderen Computer zu kopieren. Dabei haben allerdings nur der Betreiber und die von ihm autorisierten Personen ein Schreibrecht auf der Seite.

23 Das Linux-Projekt ist für eine im Bereich der freien Softwareentwicklung verhältnismäßig starke Hierarchie bekannt. Bei der Diskussion unseres Fallbeispiels KMail werden wir weiter unten sehen, daß freie Softwareentwicklungsprojekte mit weniger prominenten Projektmaintainern über andere Mechanismen der Entscheidungsfindung verfügen.

zur Verfügung zu stellen, wodurch sich der ‚Ort' der Programmentwicklung durchaus verlagern kann. Anders formuliert bleibt die Sonderrolle Torvalds' gebunden an die Akzeptanz anderer Personen.

Der zweite Faktor, der Torvalds Kontrolle über die Entwicklung des Systems verschafft, ist seine intime Kenntnis des Programms. Diese Kompetenz macht er dafür verantwortlich, daß sich andere Personen über den von ihm eingeschlagenen Weg und der verfolgten Zielsetzung noch nicht beschwert hätten. Ebenso wie die Verfügungsgewalt über die FTP-Site handelt es sich bei diesem distinktionsstiftenden Faktor um einen, der – seine Generalisierung an dieser Stelle einmal vorausgesetzt – zu einem wenig robusten Rollengefüge innerhalb von Projekten führt. Es ist wenig stabil, da es erstens voraussetzt, daß sich sämtliche Mitglieder an diesem Faktor orientieren und über ein ähnliches Verständnis von Kompetenz und Kompetenzdifferenzen verfügen. Weitere Instabilität ergibt sich zweitens daraus, daß sich Kompetenzgefälle im Verlaufe der Zeit verändern, nivellieren oder umkehren können.[24]

Ein letzter Baustein der Innovation ‚offene Programmierprojekte' wird vom Autor im Fortgang der Mail angesprochen und in der folgenden Zeit schnell realisiert: Die Gründung einer Mailingliste, auf der die Linux-Programmierer die Entwicklung des Programms vorantreiben, Entscheidungen treffen und ihre Arbeit koordinieren. Und dies unter den Rahmenbedingungen einer Offenheit der Teilnehmerschaft und der Erreichbarkeit des Orts der Entwicklung für jedermann.

Diesen Argumentationsschritt abschließend wollen wir die Innovation offener Programmierprojekte in Abgrenzung zum Software Engineering wie folgt bestimmen: Offene Programmierprojekte brechen mit den durch Software Engineering institutionalisierten normativen Vorgaben hinsichtlich des Softwareentwicklungsprozesses. Während dieser in sozialer, sachlicher und zeitlicher Hinsicht notwendigerweise stark strukturiert werden muß, wird dies zumindest in der sozialen Dimension bezweifelt und statt dessen die Entwicklung unter weitgehendem Verzicht auf formale Rollen zu betreiben versucht. Eine Rollenstruktur soll im Zuge der Entwicklung selbstorganisiert emergieren, wobei Rollendifferenzen letztlich nicht auf einer unterschiedlichen Verteilung von Nutzungsrechten am Programm und einem restriktiven Umgang mit dem Modifikationsrecht basieren, sondern im Laufe der Entwicklungstätigkeit durch Kompetenzdifferenzen entstehen sollen. Dieses laisser-faire-Prinzip findet sich auch hinsichtlich des Grades der Offenheit der Entwicklung von Software wieder. Fand die Entwicklung bis dato nicht nur

24 Weiter unten wird uns dieses kompetenzbasierte Modell der Entscheidungsfindung noch ausführlicher beschäftigen.

im Bereich der proprietären, sondern auch im Bereich der freien Softwareentwicklung in Distanz zu anderen Entwicklern und Anwendern statt, werden die Entwickler durch die Innovation der offenen Programmierprojekte mit ihren Mailinglisten und Newsgroups nun öffentlich sichtbar und für jedermann erreichbar. Mit dieser Neuerung wird das in freien Softwarelizenzen (wie der GNU GPL) angelegte Potential der Offenheit erst realisiert: Während Lizenzen das Recht zur Modifikation des Quellcodes generalisieren, schafft die Innovation der offenen Programmierprojekte die Voraussetzung, daß der Ort der Entwicklung nun auch für sämtliche interessierte Personen erreichbar wird.

Freie Softwareentwicklung und Reputation

Mit der Analyse der Diskussion zwischen Tanenbaum und Torvalds haben wir die Innovation ‚offener Programmierprojekte' mit dessen wesentlichen Merkmalen im Unterschied zu den normativen Vorgaben des Software Engineering herausgearbeitet. Die Bestimmung der Innovation war dabei bislang weitgehend negativ. Wir haben gesehen, daß das hierarchische Modell der Projektsteuerung abgelehnt wird – weitgehend unbestimmt geblieben ist dagegen, was an diese Stelle tritt. In diesem Abschnitt wollen wir nun zeigen, daß in der freien Softwareentwicklung ein normativer Orientierungshorizont von Bedeutung ist, der in einem anderen sozialen Bereich – der Wissenschaft – gut bekannt ist. In beiden sozialen Strukturen stellen ein Ethos und ein Reputationssystem diejenigen Mechanismen dar, an denen sich das Handeln orientiert, das im Fall des Wissenschaftssystems zu einer Vergrößerung von wahrheitsfähigem Wissen, im Fall von offenen Programmierprojekten zur Softwareentwicklung unter den oben beschriebenen Bedingungen führt. Als Material dienen uns hierzu erneut die Tanenbaum/Torvalds Diskussion und andere im Bereich der freien Softwareentwicklung prominente Texte. Ausgangspunkt bildet für uns ein knapper Exkurs zu Mertons „The normative Structure of Science" (1942), mit dem wir die für das klassische Wissenschaftssystem konstitutiven Normen darstellen wollen.

Mit ‚Reputation' wird üblicherweise an erster Stelle das System der modernen Wissenschaft in Verbindung gebracht. Es bildet dort ein wichtiges Medium der Steuerung von Aktivitäten, die dem Ziel einer Vergrößerung des wahrheitsfähigen Wissens dienen. Mit den normativen Grundlagen der Wissenschaft beschäftigt sich Robert K. Merton, der die institutionellen Normen der Wissenschaft in Form eines wissenschaftlichen Ethos zusammenfaßt (Merton 1942, 1957, 1972). Leitende Idee ist bei ihm, daß innerhalb der Wissenschaft ein Set von Normen existiert,

welches für die Vergrößerung des Bestands von wissenschaftlichem Wissen funktional ist. Das wissenschaftliche Ethos setzt sich aus den Normen Universalismus, Kommunismus, Uninteressiertheit und organisierten Skeptizismus zusammen. Obwohl Merton diese Normen vorrangig anhand von Texten herausgearbeitet hat, die eher selbstbeschreibenden Charakter besitzen, geht er davon aus, daß sie handlungsleitend sind und daß sich wissenschaftliches Handeln an den Normen orientiert.

Unter Universalismus versteht Merton die Norm, wissenschaftliche Ergebnisse und Befunde ausschließlich in Hinblick auf ihre Wahrheitsfähigkeit und nicht mit Blick auf die persönlichen Eigenschaften des Forschers zu beurteilen. Wahrheitsansprüche sollen unabhängig von der Person, die diese erhebt, überprüft werden, ob sie sich mit vorangegangenen Beobachtungen und anderem als validiert geltendem Wissen in Einklang befinden (ders. 1972: 48). Unter der zweiten, etwas mißverständlich bezeichneten Norm ‚Kommunismus' versteht der Autor nicht die Kollektivierung von Gütern oder Produktionsmitteln, sondern die Norm, den Stand wissenschaftlichen Wissens als ein Ergebnis gemeinsamer Anstrengung und kollektiver Leistung zu betrachten. Einzelne Beträge zum wissenschaftlichen Fortschritt werden zwar personal zugerechnet, der kollektive Ursprung von neuem Wissen, das in vielfacher Weise auf vorangehende Leistungen zurückgreift, wird jedoch betont und der Bestand von Wissen wird jedermann zugänglich gemacht: „The substantive findings of science are a product of social collaboration and are assigned to the community. They constitute a common heritage in which the equity of the individual producer is severly limited." (ders. 1942: 273) Die dritte Norm des wissenschaftlichen Ethos bildet die Uninteressiertheit (desinterestedness) des Forschers. Hierunter ist die Erwartung zu verstehen, eine Beschäftigung mit einem Forschungsgegenstand solle dem intrinsischen Interesse, nicht aber externen Anreizen – wie etwa der wirtschaftlichen Verwertung des Wissens oder persönlichen Karriereerwägungen – entspringen (ebd. 276).[25] Vierter Bestandteil bildet der organisierte Skeptizismus, worunter eine kritische Haltung gegenüber Glaubenshaltungen und Überzeugungen aufgrund ‚empirischer und logischer Kriterien' (Merton 1972: 55) verstanden wird. Damit wird eine rationale (im Sinne von begründete) Haltung als normative Leitvorstellung postuliert sowie Zurückhaltung mit dem Urteil gefordert, bis ausreichend empirische Evidenz für eine Wahrheitsbehauptung erbracht ist. Wichtiges Motivations- und Belohnungsmittel des Wissenschaftssystems bildet Reputation, also die Anerkennung und Wertschätzung von Leistungen, die diesen Normen entsprechen (Merton 1957:

25 Vgl. auch Storer 1972: 64.

297). Sie wird von anderen Experten eines Fachgebiets vergeben und findet ihren Ausdruck in der Vergabe von Positionen und Preisen sowie in Festschriften, aber auch in Zitationen in anderen wissenschaftlichen Texten. Durch Reputation wird Merton zufolge das Ziel der Wissenschaft – die Vergrößerung von Wissen – und die individuelle Motivation zusammengebunden (ebd. 323). Der Zusammenhang zwischen der Publikation von Ergebnissen und Belohnung vermittels Reputation wird auch von Hagstrom hervorgehoben. Ein Beitrag zum Bestand an öffentlich zugänglichem Wissen bezeichnet er als Geschenk (gift), das eine Gegenleistung nach sich zu ziehen hat (Hagstrom 1965: 16). Die Gegenleistung für das ‚Aufschreiben der Ergebnisse', eine Tätigkeit, die bei den von ihm beforschten Wissenschaftlern als unbeliebt gilt, bildet die Anerkennung durch Fachkollegen (ders. 1982: 25).

In diesem knappen Exkurs zu Mertons wissenschaftlichem Ethos sollte deutlich geworden sein, daß die reputationsregelnden Normen funktional sind für die Vergrößerung wissenschaftlichen Wissens. Die Absehung von den persönlichen Eigenschaften des Wissenschaftlers ist funktional für eine unvoreingenommene Prüfung seiner Ergebnisse; die Behandlung von wissenschaftlichem Wissen als kollektives Gut ermöglicht es, auf vorangegangener Arbeit aufzubauen; der organisierte Skeptizismus und die Norm der Uninteressiertheit sind funktional für die Freihaltung wissenschaftlicher Ergebnisse von verfälschenden politischen, wirtschaftlichen und ideologischen Einflüssen. Mertons Konzept des wissenschaftlichen Ethos hat eine große Anzahl von empirischen Untersuchungen angestoßen und eine Vielzahl von Kritiken evoziert. Ein erster Komplex von Kritiken bezieht sich auf die unterstellte handlungsleitende Wirkung von Normen. So relativiert Merton selbst beispielsweise mit seinen Untersuchungen zum Matthäus-Effekt (1968) die handlungsleitende Kraft der Universalismusnorm. Diesen Untersuchungen zufolge werden Arbeiten von anerkannten Personen schneller zitiert als die Arbeiten weniger bekannter Fachkollegen, während Personen mit niedrigerer Reputation von der Ko-Autorenschaft anerkannter Personen zwar profitieren (Cole 1972: 182), der überwiegende Teil der Reputation bei einer Gemeinschaftsleistung aber der im höheren Maße anerkannten Person zugeschrieben wird. Diese Arbeiten stützen die These, daß das Wissenschaftssystem über Selektionskriterien verfügt, die sich nicht ausschließlich auf die aktuellen Leistungen beziehen, sondern Wissenschaftlern auch Beachtung und Aufmerksamkeit aufgrund von vorangegangenen Leistungen zukommen.[26]

26 Luhmann unterstreicht die Selektionsfunktion von Reputation (Luhmann 1970: 237 f.). Sie konzentriert Aufmerksamkeit auf das, was mit hoher Wahrscheinlichkeit mehr Beachtung verdient als anderes (ders. 1990:

Während sich die Diskussion um die Abhängigkeit der Anerkennung wissenschaftlicher Leistungen von soziodemografischen Merkmalen auf das Spannungsverhältnis von postulierter Norm und faktischer Normenbefolgung bezieht, wird von anderen Autoren deren Konzeption kritisiert. So wenden Barnes und Dolby ein, die Normen seien eine Abstraktionsstufe zu hoch angesiedelt, empirisch nicht nachzuweisen und gehen an den wesentlichen Merkmalen der wissenschaftlichen Tätigkeit vorbei (Barnes/Dolby 1972: 270). Anstatt auf allgemeine, für das Wissenschaftssystem universell gültige Normen zu rekurrieren, schlagen sie daher in Anschluß an Kuhn (1973) vor, Paradigmen als Ort der Sozialisation von handlungsleitenden Normen zu konzipieren.[27]

Eine dritte Kritik – die sich hier allerdings gegen Mertons Schüler Hagstrom richtet – bezieht sich auf die Unvollständigkeit der Erklärung des Reputationssystems. So werde bei Hagstrom mit ‚Veröffentlichung von Forschungsergebnissen durch das Streben nach Reputation‘ zwar ein Motiv angegeben, weswegen es zu einer Publikation von Forschungsergebnissen kommt, offen bleibe jedoch, warum Ergebnisse überhaupt zur Kenntnis genommen werden (Latour/Woolgar 1982: 40). Die Autoren schlagen daher vor, das Reputationsmodell auf der Nachfrageseite durch die Annahme zu ergänzen, daß die Kenntnisnahme von Publikationen anderer Wissenschaftler die eigene Forschung verbessert und damit die Chancen des Reputationserwerbs, der Akquise von Forschungsmitteln und der Karriere erhöht (ebd. 41).

Wir wollen an dieser Stelle die Auseinandersetzung mit dem mertonianischen Modell des wissenschaftlichen Ethos abschließen und festhalten, daß dieses eine wichtige Grundlage für die Erforschung der Wissenschaftsstruktur des 19. und frühen 20. Jahrhunderts darstellt, im Zuge der Vergesellschaftung der Wissenschaft und einer stärkeren Anwendungsorientierung jedoch an Einfluß verloren hat.[28] Darüber hinaus hat

245), wobei die Kosten eines solchen Kriteriums in der Gefahr des Übersehens von wichtigen Publikationen liegen. Cole geht hingegen der Frage nach, inwieweit soziodemographische Faktoren wie Geschlecht oder Rassenzugehörigkeit die Anerkennung von Forschungsergebnissen beeinflussen. Obwohl er in seinen statistischen Untersuchungen keine Benachteiligung bestimmter Gruppen nachweisen kann, stellt er fest, dieser Befund widerspreche stark seinen Alltagserfahrungen im Wissenschaftsbetrieb (Cole 1972: 172).

27 Dementsprechend hängt dieser Perspektive folgend der Grad an Skeptizismus, den Wissenschaftler neuem Wissen entgegenbringen, davon ab, inwieweit es paradigmen-kompatibel ist (Barnes/Dolby 1972: 271).

28 Dies gilt vermutlich insbesondere für die Norm ‚desinterestedness‘, wie sich beispielsweise in Kontroversen zwischen wissenschaftlichen Experten in der Politikberatung zeigt. Die Verwissenschaftlichung der Politik durch die Nutzung von Expertise innerhalb von politischen Entscheidungspro-

die Diskussion des Normenmodells gezeigt, daß Differenzen zwischen der Normgeltung und -befolgung auftreten können, diese aber nicht die Existenz der Normen in Frage stellen. Normen zeichnen sich gerade dadurch aus, daß sie auch kontrafaktisch durchgehalten werden und Normenverstöße typischerweise zu Konflikten führen.

Während das wissenschaftliche Ethos im Bereich der Wissenschaft durch Wandlungsprozesse hin zu einer stärkeren Anwendungsorientierung in der Tendenz an Erklärungskraft verliert, da wissenschaftsexterne Normen für die Handlungsorientierung an Relevanz gewinnen, wollen wir uns im folgenden der freien Softwareentwicklung zuwenden und zeigen, daß das Modell, obwohl nicht an diesem Gegenstandsbereich entwickelt, für dieses Feld von großer Aktualität und Bedeutung ist. Zunächst wollen wir zeigen, daß Reputation unter den Entwicklern freier Software eine wichtige Kategorie darstellt und Leistungen zu deren Verleihung führen. Deutlich zeigt sich dies in der Tanenbaum/Torvalds Diskussion bei den Antipoden, deren Leistung von anderen Mitgliedern der Newsgroup hervorgehoben wird. „Dr. Tanenbaum claims that the microkernel architectur is the way to go. He has a great deal more experience with operating systems than I have. It's an understatement that it's likely that there's some substance to his statement." (Brown, K.: 3 Feb. 92.: 05:12:58) Von Interesse sind für uns hier nicht die technischen Details der Diskussion um ein angemessenes Betriebssystemdesign, sondern die Art und Weise, wie Brown es begründet, daß die Position von Tanenbaum beachtenswert ist. Nach der Nennung des akademischen Grads paraphrasiert der Teilnehmer Tanenbaums Position, um daran anschließend auf dessen Erfahrungen hinzuweisen. Die in der Vergangenheit erbrachten Leistungen werden hier in Verbindung mit der Person erinnert und genutzt, um Kompetenz zu unterstellen. Dieses Muster findet sich nun auch im Fall von Linus Torvalds. Hier führt ein anderes Mitglied der Newsgroup aus: „32 bit home computers have been available for almost a decade and Linus was the first person to ever write a working OS for them, that can be used without paying AT&T $100.000." (Foard, L.C.: 5 Feb. 92 14:56:30) Rufstiftend ist hier die Leistung von Torvalds, ein Betriebssystem für 32bit-Computer geschrieben und anderen Personen kostenlos zur Verfügung gestellt zu haben. Analog zu Tanenbaum wird hier Reputation durch die ‚Zitation' von Leistungen vergeben, aber es finden sich auch Differenzen: Erstens fällt auf, daß der Ton dieses Beitrags informeller gehalten ist, indem Torvalds bei seinem Vornamen genannt und die bei proprietären Programmen anfallenden

zessen führt auf seiner Kehrseite zu einer Politisierung der Wissenschaft (Weingart 1983, 1999, 2001), ein Prozeß, der die Frage aufwirft, inwieweit die Norm in diesem Fällen noch Orientierungskraft besitzt.

Lizenzgebühren ironisch-übertreibend beziffert wird.[29] Zweitens handelt es sich nicht nur um eine fachliche Leistung; vielmehr wird im Zitat auch hervorgehoben, dass er die normative Erwartung der Gabenökonomie erfüllt, indem er daß System zur Nutzung ‚frei' zur Verfügung stellt. Neben der fachlichen Reputation verfügt Torvalds also auch über moralische Autorität, die er sich durch diese wertvolle Gabe erworben hat.

Auch das Reputationssystem der Entwickler freier Software basiert auf Normen, die regeln, welche Leistungen anerkennungswürdig sind. Eine erste Norm tritt in der folgenden Passage deutlich hervor, in der ein Teilnehmer auf den oben zitierten Beitrag reagiert, in der ein Teilnehmer Reputation an Tanenbaum vergibt.

„>Dr. Tanenbaum claims that the microkernel architekture is the way to go.
>He has a great deal more experience with operating systems than I have.
>It's an understatement that it's likely that there's some substance to
>his statement. :-)

I tend to prefer seeing for myself rather than accepting ‚expert' opinion." (Foard, L. C. 6. Feb. 1992 09:22:40)

Die Haltung des zitierten Diskussionsteilnehmers, aufgrund der Reputation einer Person dessen Position zu übernehmen, wird hier als inakzeptabel und als Verstoß gegen das Prinzip einer eigenständigen Überprüfung einer Position gekennzeichnet und – durch die Veröffentlichung dieses Beitrags auf der Newsgroup – als Erwartung auch an andere Teilnehmer adressiert. Expliziert wird hier nicht nur die Norm des Skeptizismus, sondern, da sie auch gegenüber einem Experten eingenommen werden soll, gleichzeitig die Norm des Universalismus: Skeptizismus solle auch vor Personen mit einer hohen Reputation nicht haltmachen, die Überprüfung des Arguments also unabhängig von den Persönlichkeitsmerkmalen erfolgen.[30]

Die Orientierung an der Norm des Skeptizismus verlangt, ebenso wie im Wissenschaftssystem, ein hohes Maß an fachlicher Kompetenz der Diskussionsteilnehmer und eine große Urteilsfähigkeit. Praktisch zeigt sich dies in unserem Material nicht nur in der Aufforderung Tanenbaums, wissenschaftliche Publikationen zum Betriebssystemdesign

---

29 Der Autor nimmt hier Bezug auf proprietäre Unix-Versionen, bei denen dem Halter der Urheberrechte – der amerikanischen Telefongesellschaft AT&T – eine Lizenzgebühr gezahlt werden mußte.

30 Die Rolle von ‚Reputation' und ‚Argumentation' bei Entscheidungen innerhalb von offenen Programmierprojekten wird uns weiter unten noch ausführlich beschäftigen.

zur Kenntnis zu nehmen, sondern auch in den Hinweisen auf andere Quellen. Beispielsweise kontert ein Teilnehmer die Aussage Tanenbaums, Linux sei stark an den 386er Prozessortyp gebunden, wie folgt: „If you looked at the source instead of believing the author, you'd realise this is not true." (Shark, J.: 31 Jan 92 13:21:44) Hier wiederholt sich das aus dem obigen Zitat bekannte Muster, Aussagen nicht aufgrund der Eigenschaften des Sprechers Glauben zu schenken, sondern sich ein Urteil auf der Basis einer eigenständigen Inspektion und Analyse des Gegenstands zu bilden.

Weitere Evidenzen dafür, daß die Norm ‚Universalismus' eine relevante Orientierung darstellt, bildet die Offenheit der Softwareentwicklungsprojekte selbst. Diese zeigt an, daß neue Teilnehmer begrüßt werden und prinzipiell mit der Möglichkeit gerechnet wird, daß eine Person unabhängig von ihren Persönlichkeitsmerkmalen interessante Beiträge zur Entwicklung leisten kann, mit denen es sich zu beschäftigen lohnt. In kodifizierter Form findet sich diese Norm im fünften Grundsatz der Open Source-Definition:

„The license must not discriminate against any person or group of persons.
Rationale: In order to get the maximum benefit from the process, the maximum diversity of persons and groups should be equally eligible to contribute to open sources. Therefore we forbid any open-source license from locking anybody out of the process."[31]

Interessant ist an diesem Zitat, daß die Nicht-Diskriminierungs-Norm nicht etwa mit grundlegenden Werten wie ‚Gerechtigkeit' oder mit Figuren einer ‚political correctness' begründet wird, sondern mit Hinweis auf die positiven Folgen der sozialen Offenheit des Entwicklungsprozesses. Unterstellt wird dabei, daß prinzipiell von jedem Beteiligten, unabhängig von seinen persönlichen Eigenschaften und seiner Herkunft, ein Beitrag stammen könne, der die Entwicklung befördert.

Auf die Norm Kommunismus müssen wir wegen der ausführlichen Beschäftigung mit freien Softwarelizenzen im vorangegangenen Kapitel nur recht knapp eingehen. Das Teilen von Arbeitsergebnissen ist mit der Regulation des generalisierten Nutzungs- und Verbreitungsrechts durch die Lizenzen institutionalisiert. Neben der Distribution und Verwendung bezieht sich bei Merton ‚Kommunismus' auch auf das Zustandekommen des Produkts, das als Ergebnis kollektiver Anstrengungen und Leistungen betrachtet wird. Dieser Aspekt findet in der Tanenbaum/Torvalds Diskussion wieder:

31 Vgl. http://www.opensource.org/docs/definition.php.

„You mention OS/360 and MS-DOG[32] as examples of bad designs as they were hardware-dependent, and I agree. But there's a big difference between these and linux: linux API[33] is portable (not due to my clever design, but due to the fact that I decided to go for a fairly-well-thought-out and tested OS: unix.)" (Torvalds, L.B. 31 Jan 1992 10:33:23)

Auch hier interessieren uns wiederum nicht die technischen Details der Diskussion, sondern die Zurechnung der Leistung, daß die ‚API' als ein wichtiger Bestandteil des Betriebssystems auch auf andere Hardwareplattformen portiert werden könne. Diese Leistung kommt Torvalds zufolge nicht ihm zu, sondern geht auf seine Entscheidung zurück, das Design von Linux an Unix anzulehnen und damit Eigenschaften eines Programms zu übernehmen, das von *anderen Personen* entwickelt wurde. Der Umstand, daß eine Selbstzurechnung der Leistung denkbar und auch durchaus naheliegend wäre, jedoch von Torvalds hier nicht gewählt wird, macht seine Orientierung an ‚Kommunismus' augenfällig.[34]

Auch die vierte von Mertons Analyse des Wissenschaftssystems her bekannte Norm ‚desinterestedness' begegnet uns in der Tanenbaum/ Torvalds-Diskussion vorrangig in Form der Person Torvalds. Dieser hat mehr sukzessiv-inkrementell als geplant ein rudimentäres Betriebssystem programmiert, das er hier vorstellt und zu dessen Weiterentwicklung er andere interessierte Personen auffordert. Indem er nicht nur auf die Zahlung einer Lizenzgebühr verzichtet, sondern es auch nicht anstrebt, die weitere Entwicklung des Programms zu kontrollieren, wird deutlich, daß seine Tätigkeit nicht monetär motiviert ist.[35] Im folgenden Zitat stellt Torvalds dann sein intrinsisches Interesse an der Tätigkeit selbst heraus: „Linux has very much been a hobby (but a serious one:

32 Gemeint ist hier das Betriebssystem MS-DOS von Microsoft, das von Softwareentwicklern wenig ernst genommen und dessen Name häufig verballhornt wird. Bezeichnungen wie ‚mess-dos', ‚mess-loss', ‚messy-dos', ‚mess-dog', ‚mess-dross', ‚mush-dos' oder auch ‚domestos' (vgl. Raymond 1998: 301) sind hier der Ausdruck einer ablehnenden Haltung gegenüber Softwareunternehmen allgemein und Microsoft im besonderen.

33 Gemeint ist das Application Programming Interface, ein Bestandteil des Betriebssystems, der definiert, wie Applikationsprogramme unterschiedliche Funktionen des Betriebssystems anfordern.

34 Daneben sei an die Nennung der Namen sämtlicher zu einem Projekt beitragender Personen auf der Webseite und dem Quellcode der Programme erinnert. Dem liegt die Überzeugung zugrunde, die Programme bilden Ergebnis eines kollektiven Prozesses.

35 Mit den Gründen für eine Beteiligung an der Entwicklung freier Software werden wir uns weiter unten ausführlicher beschäftigen. Hier wird sich klären, daß das intrinsische Interesse ein basales Motiv für eine Beteiligung darstellt, neben das allerdings noch andere Faktoren treten.

the best type) for me: I get no money for it, and it's not even part of any of my studies in the university. I've done it all on my own time, and on my own machine." (Torvalds, L.B. 1991 29. Jan 1992 23:14:26) In Frontstellung gegenüber extrinsischen Motiven beschreibt er die Entwicklung des Betriebssystems als Hobby und zwar als ein ‚ernsthaftes', das er also nicht als eine oberflächliche Beschäftigung sondern als tiefgehend, intensiv und am Ergebnis orientiert verstanden wissen möchte. Hinweise darauf, daß ‚desinterestedness' nicht nur eine Orientierung von Torvalds, sondern einen generellen Bestandteil des normativen Orientierungshorizonts der Entwickler von freier Software bildet, finden sich auch an anderen Orten, wie beispielsweise in der Definition des Begriffs ‚Hacker' im „New Hackers Dictionary":[36]

„**hacker** *n*. [originally, someone who makes furniture with an axe] 1. A person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary. 2. One who programs enthusiastically (even obsessively) or who enjoys programming rather than just theorizing about programming. 3. A person capable of appreciating hack value. 4. A person who is good at programming quickly. 5. An expert at a particular programm or who frequently does work using it or on it; as in ‚a Unix hacker'. […] 6. An expert or enthusiast of any kind. One might be an astronomy hacker for example. 7. One who enjoys the intellectual challenge of creatively overcoming or circumventing limitations. 8. [deprecated] A malicious meddler who tries the discover sensitive information by poking around. Hence *password hacker*, *network hacker*. The correct term for this sense is cracker." (Raymond 1998: 234)

Aus dieser Begriffserklärung wird deutlich, daß es sich bei einem Hakker um eine Person handelt, die hohe Kompetenzen in einem bestimmten Fachgebiet besitzt. Dies gilt für sämtliche der genannten Bedeutungen, wobei hier im Mittelpunkt hier Kompetenzen im Umgang mit Computern stehen, wie die Bedeutungen 1, 2, 4, 5 und 8 zeigen. Der Hacker ist kein Normalnutzer, der nur über die unbedingt notwendigen Grundlagen der Anwendung verfügt, sondern eine Person, die bemüht ist, seine Kompetenzen stetig zu vergrößern und zu vervollkommnen. Er geht intrinsisch motiviert und passioniert einer Tätigkeit nach und nähert sich mit einer praktisch-lösungsorientierten Einstellung Problemen. Ein Hacker erscheint als ‚Macher', der die Probleme bei den Hörnern packt und sich ihnen ausgiebig widmet. Dabei geht er konstruktiv vor und ist

36 Im „New Hackers Dictionary" sind eine Vielzahl von Slang-Ausdrücken der Hacker-Kultur zusammengetragen. Eine Online-Version findet sich auch unter http://www.outpost9.com/reference/jargon/jargon_toc.html.

geleitet durch ethische Prinzipien, wie sich in Abgrenzung zur Gruppe der Cracker zeigt: Dieser Gruppe werden ausschließlich negative Eigenschaften zugeschrieben – ihr wird die Berechtigung abgesprochen, sich mit dem Ehrentitel ‚Hacker' schmücken zu dürfen.

Die Genese des Netzwerks freier Softwareentwicklung: Zusammenfassung

Ausgangspunkt dieses Teils der Arbeit war die Frage nach dem Entstehen der Netzwerkstruktur, in der freie Software entwickelt wird. Beantwortet haben wir sie durch zwei Faktoren, die nicht nur für die Genese des Netzwerks verantwortlich sind, sondern aus denen gleichzeitig einige für diese Form der Softwareentwicklung typische Merkmale resultieren: die vertragliche Innovation freier Softwarelizenzen und die organisatorische Innovation offener Programmierprojekte.

Der von uns an erster Stelle berücksichtigte Faktor waren freie Softwarelizenzen, die sich reflexiv auf das Urheberrecht beziehen. Den innovativen Gehalt dieser Verträge haben wir bestimmt, indem wir die mit der Institution des Urheberrechts verbundene Perspektive auf den Zusammenhang von Werk, Werkschöpfer und Werknutzung mit der Perspektive freier Softwarelizenzen kontrastiert haben. Dabei hat sich gezeigt, daß die Notwendigkeit einer Förderung von schöpferischer Tätigkeit – die einen zentralen Stellenwert im Urheberrecht einnimmt – aus der mit freien Softwarelizenzen institutionalisierten Perspektive zumindest für den Bereich der Softwareentwicklung bestritten und die Monopolisierung von Werknutzungsrechten zugunsten des Autors nicht als Mittel der Innovationsförderung wahrgenommen wird. Im Gegenteil: Die Setzung selektiver Anreize durch die Monopolisierung von Werknutzungsrechten und eine restriktive Vergabe derselben gilt als Innovationshemmnis und führt zu einer Vielzahl negativer Auswirkungen auf die Softwareentwicklung, die Gruppe der Anwender sowie die Beziehungen zwischen den Entwicklern von Software. Der innovative Aspekt liegt also zum einen in der Entwicklung einer neuartigen ‚Theorie' über

den Zusammenhang von Werk, Werkschöpfer und Werknutzung. Zum anderen greifen freie Softwarelizenzen auf die monopolisierten Werknutzungsrechte zurück, um diese zu generalisieren. So werden jedem Lizenzempfänger weitgehende Nutzungsrechte eingeräumt, die eine uneingeschränkte Anwendung des Programms, die Vervielfältigung, die Verbreitung und die Modifikation umfassen. Der innovative Aspekt auf der Ebene der Regulation liegt also darin, daß der vom Urheberrecht hergestellte Rechtszustand in sein Gegenteil verkehrt wird. Die wesentlichen Folgen der Lizenzen sind, daß freie Software durch die Generalisierung der Nutzungsrechte faktisch zum Kollektivgut wird, das sich aufgrund der leichten Vervielfältigbarkeit durch nicht-konkurrierenden Konsum auszeichnet – damit steht sie jedermann in prinzipiell unbegrenztem Umfang zur Verfügung. Wichtiger Effekt von über eine sogenannte Repetitionsklausel verfügenden freien Softwarelizenzen ist dabei, daß sie Beiträge zur Entwicklung ausschließlich in Form von Gaben zulassen: Sie erzwingen die Anwendung der Lizenz auch auf modifizierte Versionen des Programms, die so ohne Gegenleistungen anderen Lizenzempfängern zur Verfügung gestellt werden müssen. Dadurch ist es entweder möglich, sich in Form von Gaben an der Entwicklung des Programms zu beteiligen, oder, bei Nichteinverständnis mit den Lizenzbedingungen, Weiterentwicklungen zu unterlassen. Eine dritte Möglichkeit fällt aus. In der Summe führt diese Regulation zur Institutionalisierung einer Gabenökonomie, die mit einer einzigen Norm – der Verpflichtung zur Gegengabe – auskommt. Daneben sorgen freie Softwarelizenzen für eine sachliche und soziale Ausweitung der Möglichkeiten zur Weiterentwicklung: Die sachliche Ausweitung zeigt sich vorrangig darin, daß Formen der Entwicklung, wie die Weiterentwicklung desselben Programms, die Nutzung des gesamten Programms oder von Programmteilen zur Entwicklung anderer Programme, gestattet sind, ohne dafür die Erlaubnis des oder der ursprünglichen Autoren einholen zu müssen. Die Ausweitung in der sozialen Dimension kommt dagegen darin zum Ausdruck, daß sich jedermann an der Entwicklung beteiligen kann, da er als Lizenzempfänger bereits über das Modifikationsrecht verfügt.

Für die Entstehung des Netzwerks freier Softwareentwicklung von ebenso maßgeblicher Bedeutung ist die organisatorische Innovation offener Programmierprojekte, deren innovativen Charakter wir in Abgrenzung zu bis dato existierenden Vorstellungen über die Organisation der Entwicklungstätigkeit bestimmt haben. Für das Software Engineering haben wir gezeigt, daß der programmatische Anspruch dieser aus einer Krise der Entwicklungspraxis heraus entstandenen Disziplin in der Formalisierung der Programmentwicklung liegt. Die in der Tanenbaum/Torvalds Diskussion erstmalig auftauchende Innovation der offenen Pro-

grammierprojekte bezieht sich ausschließlich negativ auf die normativen Modelle des Software Engineering: Hier wird das Modell einer geplanten und weitgehend formalisierten Praxis verworfen und eine Entwicklung ohne formale Rollen, klare Entscheidungskompetenzen und Zurechnungsregeln für Verantwortung versucht, eine Idee, die aus der Perspektive des Software Engineering undenkbar ist. Kernbestandteil der Innovation offener Programmierprojekte bildet die Einrichtung einer öffentlich zugänglichen Newsgroup, die für sämtliche an der Programmentwicklung interessierten Personen erreichbar ist und auf der nicht nur über die Entwicklung informiert wird, sondern auch Entscheidungen über die künftigen Entwicklungswege getroffen werden.

An diesem Punkt stellt sich die Frage, was bei offenen Programmierprojekten an die Stelle des hierarchischen Modus der Handlungskoordination tritt. Ohne diese Frage hier abschließend beantworten zu können, haben wir anhand der Tanenbaum/Torvalds Diskussion gezeigt, daß unter den Entwicklern von freier Software ein Reputationssystem existiert, das große Parallelen zum Wissenschaftssystem aufweist und auf einem Ethos basiert. Kern dieses ‚Hackerethos' bilden die Normen Universalismus, Kommunismus, Uninteressiertheit und Skeptizismus, welche die Vergabe von Reputation regeln. Damit ist auf die Existenz eines Mechanismus zur Handlungskoordination in freien Softwareentwicklungsprojekten hingewiesen, jedoch seine Bedeutung noch nicht bestimmt. Wir fassen die Folgen dieser organisatorischen Innovation für die Entwicklung von Software zusammen: Während die Innovation der freien Softwarelizenzen jedermann das Recht einräumt, Modifikationen an freier Software vorzunehmen, versammelt die organisatorische Innovation der offenen Programmierprojekte sämtliche interessierte Personen an einem für jedermann zu erreichendem ‚Ort'. Offene Programmierprojekte können als Innovation verstanden werden, die das in freien Softwarelizenzen angelegte Potential einer kooperativen Programmentwicklung realisiert: Während die erste Innovation dafür sorgt, daß jedermann sich an der Entwicklung beteiligen *darf*, sorgt die Erreichbarkeit von offenen Programmierprojekten dafür, daß es auch jedermann *kann* – zumindest wenn er über ausreichende Kompetenz verfügt und bereit ist, seine Beiträge in Form von Gaben anderen Personen zu überlassen. Hiermit haben wir die Genese des Entwicklungsnetzwerks geklärt, aber allerdings gleichzeitig ein neues Bündel an Fragen gewonnen: Wie gelingt unter den beschriebenen Bedingungen eine Koordination auf ein Ziel hin? Welche Rolle spielt Reputation bei der Handlungskoordination? Existieren Mechanismen, die zur sozialen Schließung von offenen Programmierprojekten führen? Diese Fragen werden uns im folgenden beschäftigen.

Teil 2: Das Fallbeispiel ‚kmail‘

Einleitung

In Mittelpunkt dieses Teils der Studie steht die zweite leitende Frage, die nach der Organisation des Softwareentwicklungsprozesses, den Mechanismen der Handlungskoordination sowie dem Zusammenhang zwischen sozialen Strukturen und Gestalt von Technik. Dieses komplexe Fragenbündel werden wir in mehrere Teilfragen zerlegen und separat diskutieren. Im Anschluß an dieses einleitende Kapitel beginnen wir mit der Analyse der Gründe für eine Beteiligung an freien Softwareentwicklungsprojekten. Ausgehend von den innerhalb dieser Untersuchung durchgeführten Interviews werden wir die große Anzahl an Motiven herausarbeiten, die Beteiligung begünstigen und dafür verantwortlich sind, daß offene Programmierprojekte keine teilnehmerlosen Unternehmungen darstellen. In Abgrenzung zur ‚Motivationsforschung‘, die anschließend an mehr oder minder explizite Rational-Choice-Modelle nach dem Nutzen für die Akteure fragt, werden wir zeigen, daß eine solche Erklärung zu kurz greift und statt dessen vorschlagen, die Beteiligung durch institutionalisierte normative Orientierungen zu erklären.

Die leitende Frage des anschließenden Kapitels fällt in den Themenkomplex ‚Handlungskoordination innerhalb von offenen Programmierprojekten‘, in dem ein Blick auf die ‚Stunde Null‘ geworfen und der Frage nachgegangen wird, wie es zur einer Gründung kommt und welche Aufgaben dabei zu bewältigen sind. Neben einer zumindest vorläufigen Klärung der Frage, *was* entwickelt werden soll, stellt sich hier das Problem des Aufbaus einer sozialen Projektstruktur, mit der die Entwicklungstätigkeit arbeitsteilig bewältigt werden kann. Bei näherer Be-

trachtung wird sich zeigen, daß diese Anforderungen an die Gründung des Projekts widersprüchlich sind und die Technik der Modularisierung des Quellcodes eine Reaktion hierauf darstellt. Die aus der sozialen Struktur entspringenden Anforderungen führen zu einer bestimmten Gestalt von Technik, ein Befund, den wir abschließend mit Erkenntnissen der Technikgeneseforschung kontrastieren werden.

Hiernach taucht die Frage nach der sozialen Ordnung und der Handlungskoordination in offenen Programmierprojekten wieder auf – mit einer anderen Schwerpunktsetzung: Im Mittelpunkt des Kapitels ‚Entscheidung über Zielsetzungen' steht die Frage, wie den Akteuren eine Verständigung auf gemeinsam verfolgte Ziele unter den Bedingungen des Fehlens einer formalen Hierarchie, niedrigen Ein- und Ausstiegsschwellen und der Abwesenheit von monetären Anreizen gelingt. Der Normalmodus der Entscheidungsfindung bildet in offenen Programmierprojekten die Argumentation, die aber nicht immer zu Einvernehmen führen muß. Projekte verfügen daneben über einige typische Muster der Konfliktbewältigung, die wir herausarbeiten werden. Abschließen werden wir diese Überlegungen mit der Klärung des Einflusses von Anwendern auf Entscheidungen über Zielsetzungen.

Im abschließenden Kapitel gehen wir der Frage nach, wie die immer wieder betonte hohe Stabilität und Zuverlässigkeit von freier Software zu erklären ist. Ebenso wie im Fall der Modularisierung und dem hohen Maße an Anpaßbarkeit der Software bilden diese Merkmale Resultate der sozialen Struktur, in der freie Software entwickelt wird. Hier werden wir den Begriff des Realexperiments einführen und zeigen, daß die Verbreitung von freier Software als Praxis zu interpretieren ist, die auf die Gewinnung von Wissen über die Verhaltensweisen des Programms in verschiedenartigen Anwendungssituationen zielt. Die Rückkopplung von Anwendern und Entwicklern und die Mitteilung von Informationen über fehlerhafte Programmreaktionen bilden den Ausgangspunkt für die Durchführung weiterer Experimente, die zielgerichtet der Erkundung des Phänomens dienen und sich in der Tendenz dem Laborexperiment annähern. Unser zentrales Argument lautet dabei: Aus dem fortwährenden Durchlaufen dieser Rückkopplung resultiert – zumindest im Fall von reifen Programmen – eine hohe Robustheit und Stabilität bei einer geringen Zahl von ‚Bugs'.

Methodisches Vorgehen

An dieser Stelle ist es an der Zeit, einen knappen Blick auf das methodische Vorgehen der Untersuchung zu werfen. Diesem Teil der Arbeit

liegt eine Einzelfallstudie eines Softwareprojekts zugrunde, bei dessen Auswahl auf die Erfüllung der folgenden Kriterien geachtet wurde:

Größe des Projekts: Da freie Softwareentwicklungsprojekte hinsichtlich der Anzahl der Entwickler stark schwanken,[1] wurde bei der Auswahl darauf geachtet, daß das Projekt über mehrere Entwickler verfügt, deren Engagement über einen längeren Zeitraum anhält. Die Erfüllung dieses Kriteriums bildet eine Voraussetzung dafür, daß sich das uns interessierende Problem der Handlungskoordination in freien Softwareentwicklungsprojekten überhaupt erst stellen kann.

Art der entwickelten Software: Da wir uns insbesondere für Rückkopplungsprozesse zwischen Programmentwicklern und -anwendern interessieren, wurde darauf geachtet, daß das im Projekt entwickelte Programm eine Standardapplikation darstellt, die sich an einen Kreis von Anwendern richtet, der auch Nicht-Entwickler mit einschließt. Damit kamen Programme, welche die Entwickler für ihre Programmiertätigkeit selbst benötigen,[2] und solche, die sich an technisch hoch versierte Anwender aus professionellen Bereichen wenden,[3] nicht in Frage.

Dauer des Projekts/Erfolg: Dieses Kriterium wurde ebenfalls aufgrund unseres Interesses am Problem der Handlungskoordination und an den von freien Softwareprojekten entwickelten Lösungen formuliert. Wir vermuten, daß Projekte, die seit längerer Zeit existieren und bereits eine (zumindest von den Projektmitgliedern) als stabil erachtete Version veröffentlicht haben, offenbar erfolgreich die Handlungen der Beteiligten koordinieren.

Archivierte Mailingliste: Dieses Kriterium wurde aus forschungspraktischen Gründen gewählt. Eine öffentlich zugängliche, archivierte Mailingliste, in der sich sämtliche an die Mailingliste gesendete Beiträge finden, erlaubt eine schnelle und einfache Erhebung der Daten.

Unserer Einzelfallstudie basiert auf zwei Materialtypen, die sich zueinander komplementär verhalten: Zum einen ist das die Kommunikation unter den Projektmitgliedern auf der Projekt-Mailingliste, die über einen Zeitraum von zwölf Monaten analysiert wurde. Mit diesem Materialtypus können Koordinations-, Kommunikations- und Entscheidungsprozesse innerhalb des Projekts direkt beobachtet werden. Obwohl das

---

1 Vgl. hierzu die Untersuchung von Ghosh/Robles/Glott 2002: 19.

2 Z.B. Programmierumgebungen, Debugger, Compiler, Assember, Editoren.

3 Hierunter fallen Programme, die in Verwaltungs- oder Produktionsbereichen in Unternehmen eingesetzt werden (siehe: http://sourceforge.net/softwaremap/trove_list.php?form_cat=75), Programme aus dem Bereich wissenschaftlicher Forschung (siehe: http://sourceforge.net/softwaremap/trove_list.php?form_cat=97,), aber auch Programme wie Webserver (z.B. Apache, vgl. http://www.apache.org.

Archiv der Projekt-Mailingliste die stattfindende Kommunikation minutiös aufzeichnet, gewinnen wir durch die Analyse dieses Materials nur eine spezifische Perspektive auf das Entwicklungsgeschehen: So werden die ‚individuellen Gewißheiten' (Hofmann 1999: 197): normative Orientierungen, Strategien, Bewertungen und Einschätzungen, nicht – oder höchstens im Konfliktfall – kommuniziert und liegen der Kommunikation innerhalb des Projekts im Normalfall nicht-expliziert zugrunde. Zum anderen basiert die Untersuchung daher auf zehn leitfadengestützten Interviews, die mit beteiligten Entwicklern durchgeführt wurden. Zwei Interviewpartner sind Entwickler aus dem Projekt, dessen Mailingliste analysiert wurde, während die anderen Interviewpartner aus unterschiedlichen Projekten stammten. Diese Wahl wurde mit dem Ziel getroffen, einerseits nicht-explizierte Hintergründe der Entwickler unserer Einzelfallstudie analysieren zu können, andererseits aber die Ergebnisse auf Strukturähnlichkeit in anderen Projekten untersuchen, vergleichen und generalisieren zu können. Das Material wurde ad hoc ergänzt – Beispiele bilden hier die Webseiten und der Bug-Tracker unseres Fallbeispiels.

Die Auswertung folgt einer qualitativ-hermeneutischen Interpretation, wobei hier drei Aspekte der Interpretationstechnik besonders hervorgehoben werden sollen: Erstens wurde der Sequentialität des Materials, also dem zeitlichen Verlauf der Kommunikation, Rechnung getragen – sie wurde in der Reihenfolge ihres Auftretens analysiert. Ziel der Analyse ist es dabei, unterschiedliche Lesarten zu entwickeln und im Verlauf des weiteren Vorgehens Interpretationen auszuschließen und Evidenz für die übrigbleibende Lesart zusammenzutragen. Das Material wurde immer im Kontext seines Auftretens, also im Zusammenhang mit der vorhergehenden und folgenden Kommunikation interpretiert.[4] Zweitens wurden den Anfangssequenzen von Episoden – im Fall der Kommunikation auf der Mailingliste häufig (aber nicht immer) markiert durch eine neue Thread- oder Subject-Zeile – ein hohes Maß an Aufmerksamkeit entgegengebracht. Grund hierfür ist die Annahme der Sequenzanalyse, daß die Eingangssequenzen entscheidend sind für den weiteren Verlauf der daran anschließenden Kommunikation. Drittens wurde systematisch sowohl nach anderen Stellen im Material gesucht, die der Interpretation widersprechen, als auch nach solchen, in denen sich ähnliche Strukturen zeigen. Hiermit sollte die vorgenommene Interpretation nochmals geprüft und gleichzeitig vermieden werden, singuläre Ereignisse mit Strukturmerkmalen des Falls zu verwechseln.

4 Dies grenzt unser Vorgehen von den häufig als Inhaltsanalyse (Mayring 1995: 209 ff.) bezeichneten Interpretationstechniken ab.

Bei der folgenden Analyse unterschiedlicher Aspekte freier Softwareentwicklungsprojekte wird sich zeigen, daß sie zu großem Teil auf Ereignissen beruht, die von den beteiligten Akteuren als konflikthaft oder problematisch wahrgenommen werden. Der prominente Stellenwert von konfliktreichen Situationen ist hier zwei Gründen geschuldet: Konflikt zwingt die Akteure zur Explikation von im Regelfall als selbstverständlich vorausgesetzten Überzeugungen und Orientierungen. Dies vereinfacht nicht nur die Darstellung, sondern erweist sich als Ansatzpunkt für die soziologische Analyse als ungleich fruchtbarer als die aus der Perspektive der Akteure funktionierende Kommunikation des ‚techtalks', die für Nicht-Eingeweihte nur schwer nachvollziehbar ist.

Vorstellung des Fallbeispiels:

Unser Untersuchungsgegenstand bildet das Projekt KMail. KMail ist ein E-Mail-Client bzw. ein E-Mail-User-Agent für die Benutzeroberfläche KDE.[5] E-Mail-Clients sind Programme, mit denen elektronische Post gelesen und gesendet werden kann, die also zur Abwicklung des E-Mail-Verkehrs von Benutzern verwendet werden.[6] Sie unterstützen das POP3[7]-, IMAP[8]-Protocol zum Empfang und das SMTP-Protocol[9] zum Versenden von E-Mails. Neben der Standardfunktionalität eines E-Mail-Clients, dem Erstellen, Senden, Empfangen und Anzeigen der Post, verfügt KMail über eine große Anzahl weiterer Funktionen. Hierzu zählt die Integration der E-Mail-Verschlüsselung PGP[10], das Adreßbuch, eine Rechtschreibprüfung, ein Filtersystem und die Möglichkeit der Verwaltung mehrerer E-Mail-Konten.[11]

Auffälliges Merkmal des Projekts KMail bildet dessen Einbindung in eine mehrstufige Projekthierarchie. An deren Spitze steht das Verbundprojekt KDE, in dem die gleichnamige grafische Benutzeroberflä-

5 KDE (K Desktop Environment) ist eine weit verbreitete Benutzeroberfläche für Linux und Unix (vgl. http://kde.org).

6 Windows-Nutzern sollten die E-Mail-Clients Netscape Messenger, Microsoft Outlook, Microsoft Outlook Express oder Mozilla geläufiger sein. Eine Liste unterschiedlicher E-Mail-Clients findet sich unter http://en.wikipedia.org/wiki/Email_client.

7 Post Office Protocol Version 3.

8 Internet Message Access Protocol, ehem. Interactive Mail Access Protocol.

9 Simple Mail Transfer Protocol.

10 PGP steht für ‚Pretty Good Privacy'. Diese asymmetrische Verschlüsselung wurde ebenfalls in einem freien Softwareprojekt entwickelt.

11 Eine Übersicht über die wichtigsten Funktionalitäten des Programms findet sich unter http://kmail.kde.org/features.html.

che entwickelt wird und das weitere Verbundprojekte zusammenfaßt: Hierzu zählt KOffice, daß sich aus mehreren typischen Office-Anwendungen zusammensetzt, KDevelop mit einer Anzahl von Unterprojekten, die auf die Entwicklung von Programmierwerkzeugen zielt, und KDE-PIM (KDE-Personal Information Management). Hierbei handelt es sich um ein Programmbündel, das neben unserem Projekt KMail Dach für die Projekte KAddressBook, KOrganizer, KonsoleKalendar, KPilot, KAndy, KArm, KNotes und KAlarm bildet.[12]

Praktisch zeigt sich die Integration der Projekte nicht nur in der Distribution der Programme als Programmpaket, wie sie vom Projekt KDE selbst[13] und auch von Distributoren angeboten wird, der einheitlichen grafischen Erscheinung, die durch KDE vorgegeben wird, und einem einheitlichen Webauftritt der Projekte unter einer Toplevel-Domain, sondern auch durch die Koordination der Entwicklungstätigkeit zwischen den zusammengefaßten Projekten. Zum Ausdruck kommt dies im angestrebten Ziel einer guten Harmonie der Programme hinsichtlich ihrer Funktion und in einem für sämtliche Teilprojekte verbindlichen Entwicklungs- und Release-Plan, der Vorgaben darüber macht, zu welchem Zeitpunkt welche Ziele erreicht werden sollen und wann die Veröffentlichung einer neuen Version ansteht.[14]

Das Projekt KMail verfügt ebenso wie die meisten anderen Teilprojekte von KDE über eine eindrucksvolle technische Infrastruktur, die sich aus einer Webseite, Mailinglisten, dem Bug-Track-System und dem CVS-System zusammensetzt. Auf der *Webseite* des Projekts finden sich neben allgemeinen Informationen zum Projekt, Screenshoots des Programms, Informationen über die Programmurheber, aktuelle Informationen zum Stand des Projekts sowie der Funktionsweise des Programms KMail (Dokumentation und FAQ-Liste) auch Links zum Download des Programms. Auf der *Mailingliste* des Projekts findet die entwicklungsbezogene Kommunikation statt.[15] Die Mailingliste verfügt über ein öffentlich zugängliches Archiv, das bis zum Beginn des Jahres 2000 zurückreicht.[16] Das monatliche Mailaufkommen dieser Liste liegt zwischen ca. 550 und 1900 Beiträgen. Seit Januar 2004 wird diese Mailing-

12 Siehe http://pim.kde.org.

13 Die jeweils aktuelle stabile Version des Programms findet sich unter http://kde.org/download. KDE ist die standardmäßig eingerichtete Benutzeroberfläche der SuSE Linux-Distribution, wird aber von anderen Distributionen ebenfalls mitgeliefert.

14 Der sogenannte Development Plan für das KDE-Projekt findet sich unter http://developer.kde.org/development-versions.

15 Das Archiv der Entwickler-Mailingliste findet sich unter http://lists.kde.org/?l=kmail-devel&r=1&w=2.

16 Siehe http://lists.kde.org/?l=kmail-devel&r=1&w=2.

liste ergänzt durch eine ‚User Mailinglist‘, die sich das Projekt allerdings mit den anderen Projekten von KDE-PIM teilt. Diese dient der wechselseitigen Hilfeleistung unter den Benutzern des Programmpakets.[17] Mittels des *Bug-Track-Systems* wird das Feedback der Anwender von KMail an das Projekt gesammelt. Hierzu zählen nicht nur – wie bereits der Name des Systems nahelegt – Hinweise der Benutzer auf Programmfehler, sondern auch Wünsche von Benutzern, die sich auf die Funktionalität des Programms beziehen. Bugreports und Wünsche werden in separaten Listen geführt, die ohne Zugangsberechtigung gelesen werden können.[18] Beiträge können ebenfalls von jedermann ohne Zugangsberechtigung verfaßt werden. Die Listen dienen dem Projekt zur Sammlung von Ideen und zur Erinnerung an noch anstehende Arbeit. Neue Einträge werden auch an die Entwickler-Mailingliste geleitet, während im Fall einer Austragung – so beispielsweise bei der Beseitigung eines Programmfehlers – eine Nachricht an die Mailingliste automatisch generiert wird. Vierter wichtiger Bestandteil der technischen Infrastruktur bildet das *CVS-System* (Concurrent Version System).[19] Hierbei handelt es sich um ein Programm zur Verwaltung des Programmquellcodes, das einen zentralen Ablageort sämtlicher Dateien eines Projekts bildet, sämtliche Änderungen dokumentiert und Inkompatibilitäten oder Konflikte zwischen gleichzeitig stattfindenden Veränderungen verhindert.[20] Auch dieses wird vom gesamten Projekt KDE genutzt, wobei jedes Projekt über ein separates Unterverzeichnis verfügt. Um die aktuelle Version des Programms herunterzuladen, ist keine besondere Zugangsberechtigung notwendig, während das Schreiben in einem Verzeichnis nur Personen möglich ist, die über eine Zugangsberechtigung verfügen.

Das Projekt KMail wurde im Jahr 1997 gegründet und war bereits Bestandteil der ersten stabilen Version von KDE (1.0). Bis zum Juni 2004 leisteten insgesamt 49 Personen umfangreichere Beiträge zur Entwicklung, was zur Erwähnung der betreffenden Person als ‚Autor‘ des Programms führte. Diese Anzahl von Entwicklern erscheint auf den ersten Blick recht hoch. Der Eindruck relativiert sich, sobald man berücksichtigt, daß diese Personen zum weit überwiegenden Teil nur eine begrenzte Zeit am Projekt mitarbeiteten und das Projekt bereits seit sieben

17 Siehe http://lists.kde.org/?l=kdepim-users&r=1&b=200401&w=2.
18 http://kmail.kde.org.
19 CVS ist ebenfalls freie Software und wird projektförmig entwickelt. Die Homepage des Projekts findet sich unter http://www.cvshome.org. Die CVS-Implementation des KDE-Projekts findet sich unter: http://developer.kde.org/source.
20 Siehe hierzu weiterführend Purdy 2000.

Jahren existiert. Aufgrund der hohen Fluktuation liegt die Anzahl der zu einem Zeitpunkt aktiven Entwickler deutlich unter zehn, wobei nach Auskunft des Interviewten KD 1[21] die Zahl der regelmäßig aktiven Entwickler im April 2001 bei lediglich vier lag. Dieses Charakteristikum von überschaubarer Anzahl der Core-Entwickler und einer relativ großen Anzahl sporadisch beteiligter Entwickler ist recht typisch für freie Softwareprojekte und deckt sich mit den aus der Literatur bekannten Befunden.[22] Die Aktivitäten anderer Entwickler reichen hingegen von einer sporadischen Beteiligung bis hin zur episodenhaften Entwicklungstätigkeit, die über mehrere Monate anhalten kann, um dann zu ruhen und später eventuell wieder aufgenommen zu werden. Bei KMail handelt es sich um ein Projekt, daß von der Größe her eine Mittellage zwischen der Vielzahl von Ein- oder Zwei-Personen-Projekten und den relativ wenigen Großprojekten einnimmt.

Bevor wir mit der Analyse der Entwicklung des Programms beginnen, ist es notwendig, eine wichtige Rahmenbedingung zu analysieren, die nicht nur KMail, sondern sämtliche Softwareentwicklungsprojekte betrifft und die zum Verständnis der Dynamik freier Softwareentwicklung von maßgeblicher Bedeutung ist. Gemeint sind hier die Gründe dafür, daß Personen zu freien Softwareentwicklungsprojekten beitragen, also die Motive für eine Beteiligung. Die Frage danach werden wir im folgenden Kapitel in Form eines Exkurses klären.

---

21 Im folgenden werden Interviewpartner des Projekts KMail mit dem Kürzel ‚KD' (für KMail Developer), Entwickler, die sich an anderen Projekten beteiligt haben mit ‚D' (für Developer) bezeichnet.

22 Erstens wird in der Literatur darauf hingewiesen, daß die Anzahl an den Projekten beteiligter Entwickler relativ klein ist. Einer Untersuchung von Ghosh/Robles/Glott (2002: 19) zufolge werden 57,2 Prozent der 16.341 untersuchten Projekte von ein oder zwei Personen betrieben während in weiteren 37 Prozent zwischen zwei und zehn Entwickler aktiv sind. Dabei wird der Hauptteil der Entwicklungsarbeit von einer relativ kleinen Anzahl hoch engagierter Programmierer geleistet. So sind die 10 Prozent hoch aktiven Entwickler für 74 Prozent des Programmquellcodes verantwortlich (ebd. 14).

Beteiligung an freien Softwareprojekten

„Why I Must Write GNU
I consider that the golden rule requires that if I like a program I must share it with other people who like it. I cannot in good conscience sign a nondisclosure agreement or a software license agreement. So that I can continue to use computers without violating my principles, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free." (Stallman 1983, o.S.)

Das Zitat entstammt dem „Inital Annoucement" des GNU-Projekts aus dem Jahre 1983, das den Beginn freier Softwareentwicklung markiert. In der einleitenden rhetorischen Frage bringt der Autor Richard Stallman zum Ausdruck, er sei zur Entwicklung von GNU gezwungen: Er möchte oder will es nicht entwickeln, mit der Formulierung „must" macht er deutlich, daß die Entwicklung von freier Software für ihn eine zwingende Notwendigkeit darstellt. In der anschließenden Begründung tauchen die uns von der Analyse der Innovation freier Softwarelizenzen her bekannten Begründungsfiguren der Gabenökonomie wieder auf. Die Notwendigkeit resultiert aus der Norm der wechselseitigen Hilfeleistung und der ‚golden rule', sich selbst so zu verhalten, wie man es von anderen Personen erwartet. Das Zitat zeigt, daß der ‚erste'[1] Entwickler von freier Software normativ motiviert ist. Nicht persönliche Vorteile veran-

1 Wie weiter oben dargestellt, fand bereits in den 1960er und 1970er Jahren ein reger Austausch von selbstentwickelten Programmen statt. Mit der Feststellung, Stallman sei der erste Entwickler von freier Software, geht es uns nur darum zu markieren, daß dieser durch die Innovation freier Softwarelizenzen eine Gabenökonomie institutionalisiert hat.

lassen ihn zur Weitergabe der selbstentwickelten Software, sondern die für ihn handlungsleitenden Normen verlangen es.

Im Mittelpunkt dieses Kapitels stehen die Gründe, weswegen Programmierer sich an der Entwicklung von freier Software beteiligen. Dabei halten wir die an dieser Stelle übliche Frage, welchen Nutzen Akteure mit der Produktion des kollektiven Guts verfolgen,[2] für eine in die Irre leitende Problematisierung. Eine solche Frage stellt sich zwar aus der Perspektive einer Theorie rationalen Wahlverhaltens, nicht aber für die Akteure selbst. Dort wo diese Theorien ‚freeriding' als rational unterstellen, betreiben Programmierer die Entwicklung von Software getrieben von einem intrinsischen Interesse an der Tätigkeit selbst. Unser zentrales Argument lautet dabei: Der Aufwand und die Tätigkeit, die von Rational-Choice-Theorien der Kategorie ‚Kosten' zugeschlagen werden, gelten den Akteuren selbst nicht als solche. Dieses basale Motiv wird in einem ersten Schritt anhand der Interviews herausgearbeitet.[3]

Das intrinsische Interesse erklärt zwar, weswegen sich Personen intensiv mit der Entwicklung von Programmen beschäftigen, nicht aber, warum diese Aktivität im Rahmen von freien Softwareentwicklungsprojekten stattfindet. In einem zweiten Schritt werden wir zeigen, daß hierfür ein Bündel verschiedener Faktoren verantwortlich ist, die in unterschiedlichen Kombinationen für die einzelnen Entwickler eine Rolle spielen. Diesen Befund werden wir in einem dritten Schritt mit alternativen Erklärungen des Zustandekommens von Engagement für freie Software kontrastieren. Im Mittelpunkt steht dabei die Auseinandersetzung mit den in der Literatur verbreiteten Rational-Choice-Erklärungen, deren Schwierigkeiten und Grenzen wir im Zuge einer kritischen Würdigung aufzeigen werden. Hieran anschließend wollen wir eine institutionalistische Erklärung entwickeln und die Gründe für eine Beteiligung abschließend schematisch zusammenfassen.

‚Just for Fun'[4] – Intrinsisches Interesse an der Entwicklung von Computerprogrammen

Das grundlegende Motiv für die Entwickler von freier Software, sich mit dem Computer im allgemeinen und der Softwareentwicklung im besonderen zu beschäftigen, bildet das intrinsische Interesse an der Tätigkeit

2 Vgl. z.B. Lerner/Tirole 2000; Osterloh/Rota/Kuster (o.J.).
3 Wir greifen hier ausschließlich auf die der Untersuchung zugrunde liegenden Interviews zurück, da auf der Mailingliste des untersuchten Projekts die Gründe für eine Beteiligung nicht kommuniziert werden.
4 Torvalds/Diamond 2001.

selbst. Dieses wird in Texten mit selbstbeschreibendem Charakter, die sich an eine größere Öffentlichkeit richten, offensiv kommuniziert. Ein Beispiel hierfür bildet der in der Überschrift zitierte Titel von Torvalds Autobiographie. Als Begründung für die Entwicklung des Betriebssystems Linux nennt der Autor den Spaß am Programmieren, also ein unmittelbar hedonistisches Motiv. Diesem begegnet man häufig und es findet sich auch in sämtlichen im Rahmen dieser Untersuchung durchgeführten Interviews wieder. Ein Beispiel bildet die folgende Passage des Entwickler D 2, der seine Programmiertätigkeit in den Zusammenhang eines übergreifenden technischen Interesses stellt, das für ihn Klammer unterschiedlicher Aktivitäten darstellt: „Ja, also da kann ich mich noch erinnern, mein erstes Programm. Ich weiß nicht, also ich hatte immer schon ein großes Interesse, ein technisches Interesse. Ja, ich wollte ursprünglich mal Erfinder werden, das war so meine Vorstellung davon." (D 2) Die Programmiertätigkeit wird vom Interviewten in dieser Darstellung in den größeren Kontext seiner biographischen Entwicklung eingebettet und mit seinem Berufswunsch ‚Erfinder' in Verbindung gebracht – einer Person, die sachtechnische Einrichtungen und Apparaturen in genialer Weise erschafft und eher dem Reich der Phantasie angehört,[5] denn als tatsächliches Berufsfeld existiert. Aus welchen Bestandteilen setzt sich das intrinsische Interesse an der Computerprogrammierung nun genau zusammen? Ein wichtiger, regelmäßig auftauchender Aspekt bildet die ‚Neugierde', die in der nachstehenden Passage plastisch beschrieben wird:

„Ich stand damals vor den ersten Videospielen, die man für die Fernseher so verkauft hat. Aber nicht die allerersten, sondern so Atari 2600, logisch. Da stand ich davor und fand das spannend. Damals hat man auch Videospiele in Kaufhäusern gefunden oder so. So daß man da als Jugendlicher dran durfte. So lustige Hüpf- und Spring-Spiele gab auch schon und da hab ich gedacht, ich möchte wissen, wie so ein Ding funktioniert. Ich bin grundsätzlich neugierig. Also, auf vielen Gebieten, da war ich auch neugierig und es ist so angefangen, daß ich irgendwann mal ein Basic-Buch geschenkt bekam, was ich jetzt kei-

5 Die technischen Pioniere des ausgehenden 19. und des beginnenden 20. Jahrhunderts, die sich eingehend mit der Erforschung von technisch nutzbaren Wirkungszusammenhängen beschäftigt haben (vgl. Hughes 1991, 1985), entsprachen diesem romantischen Bild vielleicht noch eher, als die Beschäftigten in FuE-Abteilungen heutiger Unternehmen. Der Typus des Erfinders begegnet uns in den Mythen, in denen die Entstehung des Personal Computers und von IT-Unternehmen im Zuge des Computerbooms erzählt wird. Vgl. hierzu Rogers/Larsen 1986: 14 ff. zur Entstehung von Apple, McSummit/Martin 1989: 365 ff, am Beispiel von AMD, SPC (Hersteller von Office-Applikationssoftware) und Grid (Hersteller einer der ersten Laptops).

nem erzählen darf.[6] Und, ja so mit elf oder zwölf hatte ich dann das erste Mal einen Computer selber zu Hause zur Verfügung. Damals gab's noch nichts, was man mit dem Computer sonst hätte machen können, da habe ich angefangen damit rumzuspielen, rumzuprogrammieren und seitdem mache ich immer wieder Programmentwicklung." (D 7)

Die visuellen Eindrücke von hüpfenden und springenden Figuren auf einem Bildschirm bilden für den Interviewten den Ausgangspunkt für die Frage nach den dahinterstehenden Funktionsprinzipien und deren Konstruktionsweise. Diese Neugierde taucht in der Beschreibung der Umgangsweise mit dem ersten Computer und dem eher zufällig zur Verfügung stehenden Basic-Buch wieder auf: Der Computer wird vom Interviewten als wenig nützlich beschrieben, ein Umstand, der auch dazu hätte Anlaß geben können, die Beschäftigung mit ihm wieder einzustellen.[7] Der Interviewte setzt diese hingegen fort, wobei er die Art der Auseinandersetzung als ‚rumspielen' und ‚rumprogrammieren' also als wenig zielorientiert und leistungsentlastet beschreibt.

Neben dieser Befriedigung von Neugierde durch die Erkundung von Funktionsprinzipien und den Blick unter die Oberfläche stellt die Erweiterung und Vervollkommnung der eigenen Programmierfähigkeiten einen zweiten Teilaspekt des Interesses dar. „Also ich hab tausend Mal Fraktalprogramme geschrieben in meinem Leben. In verschiedensten Programmiersprachen, einfach um das mal wieder zu lernen. Weil's halt relativ nett ist, man hat dann ein schönes buntes Ergebnis nachher und es ist nicht allzu schwierig so zum Einstieg." (D 1) Die wiederholte Entwicklung desselben Programms, das Bilder bricht und umwandelt, dient hier ganz offensichtlich nicht dem Zweck der Erreichung eines bestimmten Entwicklungsergebnisses, sondern – in der Handlungskette weiter vorgelagert – dem Erwerb von Kompetenzen zur Mittelkonstruktion. Die besondere Eignung des gewählten Entwicklungsziels wird durch den Schwierigkeitsgrad und das herbeigeführte Ergebnis begründet: Der Interviewte setzt sich beim Erwerb von neuen Kompetenzen erreichbare Ziele, das bunte Ergebnis, das beim Funktionieren des Programms vom

6 Bei Basic handelt es sich um die erste populäre Programmiersprache, die allerdings von Softwareentwicklern nicht ernst genommen wurde, weil sie als Anfängersprache konzipiert war und sich im Vergleich zu Alternativen wie C durch eine geringe Mächtigkeit auszeichnete.

7 Beispiele für eine problematische Aneignung von Computern, die den Nutzer zu Umgehungs- und Vermeidungsstrategien veranlassen, finden sich bei Herrlyn 2002: 57. Im bemerkenswerten Kontrast hierzu steht die Einstellung der interviewten Entwickler, die Schwierigkeiten und Probleme als Herausforderungen interpretieren und Anlaß für eine weitergehende Beschäftigung mit dem Computer bilden.

Bildschirm ausgegeben wird, bildet Beleg und Bestätigung für den erfolgreichen Erwerb neuer Kompetenzen.

Ein dritter Aspekt, der in den Interviews als Bestandteil des intrinsischen Interesses genannt wird, bildet Könnerschaft. „Das weiß ich gar nicht mehr konkret, ich weiß nicht, einfach so generell Interesse an Computern da gehört das einfach so dazu würde ich sagen, daß man einfach auch selber was können will." (KD 2) Die Könnerschaft führt zu Befriedigung, da vermittels typischerweise autodidaktisch erworbener Kompetenzen Entwicklungsziele erreicht werden und die selbstgeschaffenen Computerprogramme zur Realisierung selbstgesetzter Handlungsziele eingesetzt werden können.

Das Phänomen des intrinsischen Interesses ist aus der Literatur zur Computernutzung gut bekannt, auch wenn es z.T. anders interpretiert wird. Ein Beispiel hierfür bildet die klassische Untersuchung „Die Macht der Computer und die Ohnmacht der Vernunft" von Joseph Weizenbaum (1977), in der er sich neben anderen Themen für das zu diesem Zeitpunkt neuartige Phänomen der Programmierer interessiert, die sich, obwohl dort nicht angestellt, in amerikanischen Rechenzentren aufhalten und programmierend ihrem Interesse nachgehen. Computerbegeisterte Programmierer (oder mit den Worten Weizenbaums: Computerfetischisten) werden hier ausschließlich negativ etikettiert, als zwanghaft bezeichnet und unterstellt, es ginge ihnen nicht um die Konstruktion eines Mittels, sondern um das Allmachterlebnis über den Computer (ebd. 166), das mit dem Verhalten krankhafter Glücksspieler zu vergleichen sei (ebd. 169 ff.). Weizenbaum bedient sich damit eines weitverbreiteten Stereotyps des sozial unfähigen, von seinen Handlungen ‚besessenen' Entwicklers und kontrastiert dessen intensive Beschäftigung mit von ihm als normal oder angemessen erachteten Formen der Auseinandersetzung mit dem Computer, so daß er zu einem alarmierenden Bild der Folgenhaftigkeit einer übertrieben intensiven Beschäftigung gelangt. Auch wenn er durch seine stark normativ aufgeladenen Beschreibungen mehr über die technik- oder computerkritische Stimmung dieser Zeit aussagt[8] als über die Programmierer selbst, kommt ihm jedoch der Ver-

8 Die Stereotypisierungen, mit der die intensive Beschäftigung mit dem Computer als pathologisch klassifiziert wird, war in den 1980er Jahren nicht nur in der psychologischen Literatur häufig anzutreffen. So beschreibt Volpert (1985: 62) die Programmierer als Zwangs- und Suchtcharaktere, die von Allmachtsphantasien angetrieben sind, während Johnson (1980: 808 f.) die Begeisterung als Flucht aus der ‚alten', realen Welt in eine ‚neue', rein symbolische Welt interpretiert, wobei die Ursache für das Verhalten in neurotischen Störungen gesehen wird. Helms (1983: 931) läßt es an einem pauschalisierenden Kulturpessimismus am Element der Computerkritik nicht fehlen, wenn er auf „kindliche und erwachsene Pro-

dienst zu, auf das Phänomen der hohen Faszination aufmerksam gemacht zu haben, die der Computer ausüben kann.

In der neueren Literatur wird das Phänomen inzwischen anders interpretiert. Das intrinsische Interesse, mit dem Programmierer z.T. ihrer Tätigkeit nachgehen, wird beispielsweise von Eckert et al. in Zusammenhang mit der Aneignungsform gerückt, die analog zu anderen Freizeitaktivitäten vorrangig spielerisch und zweckfrei durch ein trial-and-error-Prinzip und learning-by-doing geleitet stattfindet (Eckert et at. 1990: 196). In ähnliche Richtung argumentiert Braun, wenn sie die Orientierungen von Informatikern typisiert und als gemeinsames Merkmal der Typen die von einem Eigeninteresse getriebene Aneignung des Computers zu einem frühen biographischen Zeitpunkt herausarbeitet (Braun 1998: 284). Im Unterschied zu Weizenbaum vermeiden diese Autoren eine normative Bewertung des Engagements. In der Literatur zu freier Softwareentwicklung läßt sich ein hohes Maß an Einmütigkeit hinsichtlich des Stellenwerts eines intrinsischen Interesses als Motivation feststellen. So betonen Lakhani/Wolf (2003: 3), die stärkste Motivation für eine Beteiligung sei ein Interesse an der Tätigkeit und „how creative a person feels when working on the Project“, während dieses Motiv von anderen Autoren ebenfalls an prominenter Stelle genannt wird (z.B. Hertel et al. 2003; Osterloh et al. (o.J.); Tzouris 2002).

Extrinsische Gründe

Mit der Analyse des intrinsischen Interesses kann zwar erklärt werden, warum sich Programmierer mit der Entwicklung von Software beschäftigen, nicht aber, weswegen dies in zunehmendem Maße im Rahmen offener Programmierprojekte stattfindet. Um diese Frage zu beantworten, wollen wir zeigen, daß die Kooperation mit anderen Entwicklern Vorteile gegenüber einem alleinigen Vorgehen hat.[9] Auf einen ersten Vorteil macht ein Entwickler in der folgenden Interviewpassage aufmerksam:

---

grammierer“ zu sprechen kommt, die hochgradige psychische und soziale Störungen aufweisen.

9 Strenggenommen ist zu unterscheiden zwischen (a) der Entwicklung von freier Software und der Entwicklung von proprietärer Software sowie (b) der Entwicklung von Software innerhalb von offenen Programmierprojekten und der Entwicklung von Software innerhalb von anderen Organisationsformen. Wir diskutieren im folgenden die Gründe der Entwicklung von freier Software in offenen Programmierprojekten und vergleichen diese mit einer Alternative, die von sämtlichen Programmierern gewählt werden kann: Die alleinige Beschäftigung mit Softwareentwicklung. Wir wählen

„Ich hab gesehen, daß egal wie viel Zeit ich investiere, ich nie das erreichen kann, was die anderen Projekte schon erreicht haben. Wenn ich also ein anderes Projekt verbessern kann, ist das viel mehr wert, als wenn ich von vorne anfange. Also, ich kann mich jetzt irgendwie hinsetzen und einen Editor schreiben, aber das kann ich 20 Jahre lang machen und Emacs ist immer noch besser. Also, warum sollte ich das tun?“ (D 1)

Der Vorteil einer kooperativen Programmentwicklung in offenen Programmierprojekten gegenüber einem alleinigen Vorgehen liegt im Umfang der erreichbaren Ziele, der wegen des arbeitsteiligen Vorgehens und des Rückgriffs auf bereits geleistete Entwicklungsarbeit sehr hoch ausfällt.[10] Der Vergleich macht deutlich, daß der Programmierer an der Verwendbarkeit des Programms interessiert ist, auch wenn derselbe Entwickler in einer weiter oben zitierten Passage die Programmiertätigkeit auch als Mittel des Erwerbs von neuen Programmierkompetenzen darstellt. Der Erwerb von Kompetenzen bildet nicht Endzweck, sondern vielmehr Voraussetzung und Ressource, um mit anderen Personen innerhalb von offenen Programmierprojekten umfangreichere Entwicklungsziele zu verfolgen. Die Vorteile eines gemeinsamen Vorgehens erschöpfen sich nun allerdings nicht in dieser Möglichkeit, weiter gesteckte Ziele zu erreichen: „Wenn ich sage, ‚hier, Fehler im Programm‘, wird das sofort gefixt über Nacht und dann hat derjenige mir ein Update geschickt. Das macht einem dann Freude. Das kommt nicht jeden Tag vor, aber regelmäßig genug, als daß das Umfeld Spaß macht.“ (D 7, Interview) Schön zum Ausdruck bringt dieser Entwickler, daß ein Ineinandergreifen der Tätigkeit verschiedener Personen zu einem hohen Maß an Zufriedenheit mit der Entwicklungstätigkeit und zum Eindruck eines schnellen Vorankommens im Projekt führt. Die Entwickler müssen im Unterschied zum alleinigen Vorgehen nicht sämtliche Probleme und Schwierigkeiten selbst bewältigen, sondern können damit rechnen, daß andere Beteiligte einen Teil der Aufgaben erledigen. Ein drittes Motiv, das Kooperation in offenen Programmierprojekten begünstigt, bildet die Orientierung daran, etwas Nützliches zu entwickeln.

---

diesen Vergleich, da eine andere denkbare Alternative – die Entwicklung von proprietärer Software – nicht für sämtliche Entwickler verfügbar ist.

10 Allerdings ist das Emacs-Projekt sehr alt und von daher ein sehr plastisches Extrembeispiel. Der Editor wurde 1976 von Richard Stallman für die Großrechner PDP-6 und PDP-10 am MIT entwickelt, vgl. http://en.wikipedia.org/wiki/Emacs, und seit dem einige Male vollständig neu geschrieben. Die bekannteste Variante ist GNU Emacs, ebenfalls in seiner ursprünglichen Form von Stallman implementiert, siehe http://www.gnu.org/software/emacs/emacs.html.

„Gerade auch, wenn man sich da vorstellt, daß Linux jetzt irgendwie 2 Prozent Desktop Markt hat und man sich dann ausrechnet, 70 Prozent läuft unter KDE. Und wenn dann 50 Prozent von KDE[-Anwendern, N.C.T.] KMail benutzen, dann sind das Hunderttausende von Leuten, die dieses Programm benutzen. Das ist schon beeindruckend. Daß man da irgendwas geschrieben hat, was andere dann benutzen. Ein Programm, was man selber geschrieben hat, wird benutzt. Das ist schon ziemlich gut. Da kriegt man bei diesen Open-Source-Entwicklungen Motivation." (KD 1)

Die Orientierung am Erfolgskriterium der Anzahl der Nutzer des Programms bildet einen Faktor, der die Kooperation innerhalb von freien Softwareentwicklungsprojekten einerseits und den Zusammenschluß von Softwareentwicklungsprojekten zu Dachprojekten andererseits begünstigt: Die Kooperation mit anderen Entwicklern *innerhalb* des Projekts ermöglicht die Realisierung von weitergehenden Zielsetzungen, wodurch das Programm für eine größere Anzahl von Benutzern interessant wird. Die Kooperation mit anderen Projekten im Rahmen eines Projektverbunds vergrößert hingegen die Verbreitung und Bekanntheit des Programms durch die Nutzung gemeinsamer Vertriebswege: Die Programme des Dachprojekts werden gemeinsam als Programmpaket vertrieben, wodurch sämtliche Teilprogramme zur Prominenz des Gesamtprojekts beitragen, aber gleichzeitig auch von seiner Prominenz profitieren. Die Verbreitung, die es dem Entwickler erlaubt, das Produkt als nützlich und die Tätigkeit der Programmentwicklung als sinnvoll zu interpretieren, hat allerdings noch eine weitere Folge, die Kooperation attraktiv macht. Mit der Verbreitung des Programms steigt das Maß an Rückkopplung zwischen Anwendern und Entwicklern an. „‚Danke, Danke sehr', also das hat mich beim PAM-Modul[11] immer wieder motiviert. Ich hab das ja für uns[12] geschrieben, und dann hab ich es released und es waren einfach tausend Leute da, die es benutzt haben. Und 'ne Menge von denen haben irgendwie gesagt, wie praktisch es für sie ist." (D 3) Die in diesem Zitat angesprochene Anerkennung durch Anwender bildet einen Faktor, der sich stabilisierend auf die Beteiligung auswirkt. Anerkennung muß sich allerdings regelmäßig wiederholen, um zu einer zeitstabilen Quelle von Motivation zu werden, die den Entwickler dauerhaft zur Beteiligung an einem Projekt veranlaßt. Auch dieser Faktor hat die organisationale Struktur von offenen Programmierprojekten zur

11 PAM meint Plugable Authentication Module und werden von Administratoren genutzt, um Zugriffsrechte von Benutzern auf den Apache-Webserver zu verwalten.

12 Gemeint ist hier ein Unternehmen, in dem der Programmierer zum Zeitpunkt der Programmentwicklung als Administrator tätig war.

Voraussetzung. Die direkte Rückkopplung zwischen Entwicklern und Anwendern ist nur möglich, sofern die Entwickler den Anwendern bekannt und für diese erreichbar sind. Diese Bedingungen sind im Fall von offenen Programmierprojekten erfüllt, da die Mailingliste offen ist und jedermann Beiträge an sie schicken kann.[13]

Neben den strukturellen Rahmenbedingungen offener Programmierprojekte ziehen auch freie Softwarelizenzen Konsequenzen nach sich, die eine Beteiligung an der Entwicklung begünstigen. Zu nennen ist hier an erster Stelle das dort ausgesprochene Modifikationsrecht, das Voraussetzung für die Möglichkeit einer Anpassung des Programms an eigene Anwendungsbedürfnisse bildet. Der Stellenwert dieser von der Realisierung eigener Zwecksetzungen angetriebenen Entwicklungstätigkeit hat auch Eingang in Texte freier Softwareentwickler mit selbstbeschreibendem Charakter gefunden. So postuliert beispielsweise Raymond (1999: 32): „Every good work of software starts by scratching a developer's personal itch", um daraufhin festzustellen: „Necessity is the mother of invention". Beispielhaft dafür, daß eigene Anwendungsbedürfnisse Motive für die Beteiligung darstellen, sei die folgende Passage eines Entwicklers zitiert, der Veränderungen an einem Editor vornimmt, den er selbst zur Softwareentwicklung einsetzt:

„Und bei diesem Editor gibt es so'n sogenanntes Syntaxhighlighting,[14] das heißt, man schreibt da seinen Quelltext und es werden gewisse Schlüsselwörter mit 'ner anderen Farbe markiert. Kommentare, die man da reinschreibt, werden ein bißchen ausgegraut, damit die nicht so sehr hervorstechen. Und so was gab es für diese Programmiersprache halt noch nicht. […] [D]er Grund, das zu machen, ist natürlich einerseits, ich hab das noch nicht und ich will das, weil ich diesen Editor benutzen und weil ich damit schreiben will […]." (D 2)

Die Bedeutung der eigeninteressierten oder von eigenen Bedürfnissen angetriebenen Entwicklungsbeteiligung darf allerdings nicht überschätzt werden. Durch eigene Anwendungsinteressen kann erklärt werden, warum es zu einem überschaubaren Engagement von kürzerer Dauer kommt, wie es bei der Beseitigung von Bugs und der Implementation von funktionalen Erweiterungen mit überschaubarem Ausmaß der Fall ist, nicht aber, weswegen ein länger andauerndes Engagement mit hoher

13 Diese Art von Rückkopplung zwischen Anwendern und Entwicklern ist auf der Mailingliste des Projekts KMail ein weitverbreitetes Phänomen. Häufig leiten die Entwickler persönliche E-Mails, in denen Anwender dem Entwickler danken, an die Mailingliste weiter.

14 Syntaxhighlighting ist eine Standardfunktionalität, über die nahezu sämtlichen Programmeditoren verfügen. Es kann vorausgesetzt werden, daß diese den Entwicklern gut bekannt ist.

Intensität geleistet wird. Das Verhältnis zwischen dem Aufwand für die Entwicklung und dem erreichten Nutzen in Anwendungssituationen fallen stark auseinander. Ein dauerhaftes Engagement ausschließlich durch die eigene Anwendungsbedürfnisse zu erklären, hieße, von einem hohen Maß an Irrationalität des Entwicklers auszugehen, da so unterstellt wird, er verhalte sich gegenüber aufgewendeter Zeit und Ressourcen ignorant.

Neben den auf individueller Ebene zu verortenden Gründen und den auf die Rahmenbedingungen freier Softwareentwicklung zurückgehenden Faktoren finden sich in den Interviews auch Motive, die normativen Ursprungs sind. An erster Stelle zu nennen ist hier die Reziprozitätsnorm der Verpflichtung zur Gegengabe. Diese Norm taucht in einem Großteil der Interviews bei der Begründung des Engagements mehr oder minder explizit auf und zeigt sich sehr deutlich in der folgenden Interviewpassage eines Programmierers, der sich aufgrund der Möglichkeit zur Nutzung von freier Software verpflichtet fühlt, selbst eigene Beiträge zur Entwicklung beizusteuern:

„Das kann man vielleicht so ein bißchen auf ein Schuldgefühl, auf ein [betont] Schuldig-keits-gefühl zurückführen. Ich habe diese freie Software. Ich hab sie ganz umsonst gekriegt. Ich darf sie jetzt benutzen. Und, wenn ich daran was mache, weil mir daran irgendwas nicht gefällt und ich verbessere das, dann bin ich es den anderen Leuten schuldig, das ihnen auch wieder zurückzugeben. Also das ist so eine primäre Motivation.“ (D 2)

Wichtig zum Verständnis des vom Interviewten genannten Schuldigkeitsgefühls ist, daß – wie oben gezeigt – freie Software ein kollektives Gut darstellt, welches durch freie Verteilbarkeit und nicht-kon-kurrierenden Konsum gekennzeichnet ist. Aufgrund dieser Eigenschaften ist es für niemanden nachvollziehbar, ob eine Person das Gut nutzt, ohne zu dessen Produktion beigetragen zu haben. Nicht-Beteiligung an der Bereitstellung des Guts ist weder kontrollier- noch sanktionierbar. Dementsprechend bildet die Reziprozitätsnorm hier eine interne Orientierung und stellt keinen äußerlichen ‚constraint‘ dar, bei dem eine Normenbefolgung aus bloßen strategischen Gründen stattfinden würde.

Einen zweiten Typus von normativen Gründen mit stärkerer gesellschaftspolitischer Prägung haben wir bereits zu Beginn dieses Kapitels anhand des Eingangszitats von Richard M. Stallman analysiert und ist uns auch bereits bei der Beschäftigung mit den normativen Grundlagen freier Softwarelizenzen weiter oben begegnet. An dieser Stelle wollen wir mit dem folgenden etwas längeren Zitat zeigen, daß auch diese normative Orientierung einen Faktor darstellt, der sich beteiligungsfördernd auswirkt.

„Also in unserer Gesellschaft ist IT eigentlich das einflußreichste Ding, was wir haben. Es bestimmt und durchdringt unser Leben wie nie zuvor, noch keinerlei andere Sachen. Und meine Meinung ist, daß wir die Kontrolle darüber nicht abgeben sollten, was unser Leben bestimmt. Denn in der Demokratie ist eigentlich der Grundgedanke, daß wir selbstbestimmt sind. […] Wer aber in der Zukunft die IT kontrolliert und niemandem das Know-how überträgt – das Know-how also nur wenigen, die dann bestimmen können, wo's weitergeht, das halte ich für sehr antidemokratisch. […] Und – mit proprietärer Software ist es [mir, N.C.T.] nicht möglich, Kontrolle auszuüben über das, was mein Leben so stark beeinflußt. Das kann nur mit freier Software passieren. Und Software sollte eigentlich allen gehören. Das klingt sehr kommunistisch, ist eigentlich demokratisch und wirtschaftlich gedacht. Dieses Ideal umzusetzen ist ein großer Faktor, die Firma zu führen." (D 7)

An dieser Stelle geht es nicht darum zu beurteilen, ob die Aussagen zutreffend und ob die gesellschaftspolitischen Ideen zustimmungsfähig sind, oder ob der dargestellte Zusammenhang logische Widersprüche aufweist. Entscheidend ist für uns, daß die vom Entwickler eingenommene Perspektive auf den Zusammenhang von Technik und gesellschaftlichen Machtverhältnissen für sein Engagement mit bestimmend ist. Seine Tätigkeit, das Führen eines Unternehmens, ist – nicht nur dieser Erläuterung nach[15] – moralisch gebunden. Unternehmensziel bildet sowohl die Erwirtschaftung von Gewinnen als auch die Verfolgung der genannten gesellschaftspolitischen Ziele.[16] Die Gründung und die Aktivitäten des Unternehmens werden in einen komplexen Deutungszusammenhang gerückt, als Mittel zur Förderung von freier Software beschrieben, die aufgrund ihres emanzipatorischen Potentials als unterstützenswert gilt.

Ein dritter Faktor aus dem Bereich der institutionalisierten Normen, der Einfluß auf Beteiligung an freien Softwareprojekten nimmt, ist die Reputation. Wie weiter oben ausführlicher gezeigt, meint Reputation einen Ruf, der auf der Anerkennung von Leistungen durch andere Ent-

15 Das Unternehmen trägt nicht nur durch Entwicklungsbeiträge zu freier Software bei, sondern bezahlt zum Teil auch Programmierer für ihre Tätigkeit in Entwicklungsprojekten. Diese Bezahlung ist nicht an die Erledigung bestimmter Aufgaben gebunden – wie es beispielsweise bei Werkverträgen der Fall wäre –, sondern erfolgt z.T. ex post für die bisherigen Leistungen im Projekt. Damit qualifizieren sich die Beiträge als Gaben. Die dafür aufgewendeten Mittel stammen von den Kunden des Unternehmens und sind Bestandteil der entsprechenden Leistungsverträge (Informationen aus den Interviews mit D 6 und D 7).

16 Die Orientierung an den Normen der freien Softwareentwickler sollte in diesem Fall nicht verwundern, handelt es sich doch um ein Unternehmen, das noch recht jung ist und von aktiven Entwicklern gegründet wurde.

wickler (Peers) basiert. Diese motivierenden Effekte beschreibt ein Interviewpartner wie folgt:

„Jetzt auch auf der Cebit habe ich gerade die ‚Intevation'-Leute[17] aus Osnabrück, Werner Koch kennengelernt, der GnuPG[18] programmiert, und Dirk Stauch, der Source Forge[19] entwickelt hat. Und solche Leute lernt man alle kennen und einige Leute haben dann auch Respekt vor mir – im Prinzip schon [lacht]." (KD 1)

Interessant an dieser Schilderung ist für uns, daß der Interviewte die Personen namentlich vorstellt und ihre Leistungen erwähnt, also – wie es für Reputationssysteme eben typisch ist – Leistungen mit Personen verknüpft. Durch die Nennung von Personen mit hoher Reputation, die ihm mit Respekt begegnen, wird die Logik der Vergabe von Reputation deutlich. Die Anerkennung ist um so wertvoller, je höher die Reputation desjenigen ist, von dem diese stammt.

Einen letzten Faktor, der die Beteiligung fördert, bilden monetäre Anreize. Spätestens seit der Gründung von Unternehmen in diesem Bereich und dem Entstehen eines Markts für mit freier Software verknüpfte Dienstleistungen ist zu dem Bündel der bis hierher herausgearbeiteten Motive ein weiteres hinzugetreten: die Bezahlung als direkte Kompensation für Entwicklungsbeiträge. Auch in den hier geführten Interviews berichten mehrere Programmierer davon, mit ihrer Tätigkeit Geld zu verdienen. Das Spektrum reicht dabei von einer gelegenheitsmäßig betriebenen Tätigkeit bis hin zu einer festen Anstellung mit regelmäßigem Einkommen. Um den Stellenwert monetärer Anreize abschätzen zu können, ist es notwendig, sich zu vergegenwärtigen, daß die Aufnahme des Engagements in der Regel ohne eine Aussicht auf Bezahlung erfolgt und die Akteure sich Einkommensquellen meist erst im Zuge eines länger anhaltenden Engagements erschließen können.

„Also, man fängt nicht damit an, mit der Hoffnung, daß man dafür bezahlt wird. Man hat vielleicht irgendwann mal die Hoffnung, wenn man denn wirklich mal sehr, sehr viel Zeit über einen längeren Zeitraum in ein Projekt investiert, daß sich das irgendwie auszahlt, aber das ist nicht der Einstieg, weil man normalerweise mit sehr wenig anfängt." (D 2)

---

17 Gemeint sind hier die Angestellten des Unternehmens ‚Intevation', das mit freier Software verknüpfte Dienstleistungen anbietet.

18 GnuPG meint ‚GNU Privacy Guard', ebenfalls freie Software.

19 Bei SourceForge handelt es sich um eine Internetplattform, die eine Vielzahl von freien Softwareentwicklungsprojekten hosted. Siehe http://source forge.net/index.php.

Deutlich wird, daß das Engagement – meist von geringem Umfang – und nicht die Bezahlung oder die Aussicht darauf am Anfang steht. Wie verhalten sich nun aber monetäre Anreize und ein hohes Engagement zueinander? Folgt das Engagement letztlich der Bezahlung oder bildet die Bezahlung eine nicht notwendigerweise angestrebte Folge eines hohen Engagements? Diese Fragen lassen sich anhand des vorliegenden Materials nicht abschließend klären.[20] Im folgenden wollen wir jedoch vorschlagen, die Bedeutung der Bezahlung eher zurückhaltend einzuschätzen, da sie – wie wir anhand des folgenden Zitats belegen wollen – nicht durch das Engagement für freie Software angestrebt werden muß, sondern vielmehr auch eine Voraussetzung dafür darstellen kann, daß ein Engagement von hoher Intensität überhaupt möglich ist. Ein hochengagierter Entwickler, der sein Engagement für die Entwicklung von freier Software sukzessive gesteigert hat, erläutert dies wie folgt:

„Es kam auch irgendwann der Punkt, wo ich sagen mußte, ich kann mir nur die Zeit nehmen dafür,[21] wenn ich es auch schaffe, irgendwie dafür bezahlt zu werden, weil einfach die Zeit irgendwann zu Ende ist. Man hat halt 24 Stunden und wenn die weg sind, sind die weg. So, und dann ist dann irgendwann der Punkt, wo man sagen muß [betont]: ‚tja, Leute, wenn ihr wollt, daß ich das mache, dann müßt ihr mich bezahlen'." (D 8)

Das hohe Engagement geht in dieser Schilderung der Bezahlung voraus. Es wird zu Beginn ohne eine monetäre Kompensation geleistet, Bezahlung bildet also hier keine Voraussetzung dafür, daß der Programmierer seine Zeit zur Entwicklung von freier Software nutzt. Da dieses Engagement buchstäblich tagesfüllend ist, ist es nur zu leisten, sofern er sich dafür bezahlen läßt. Wir vermuten, daß dieser Fall recht typisch ist und Bezahlung häufig die Rolle der Voraussetzung für die Möglichkeit eines hohen Engagements und nicht die eines Motivationsmittels spielt, da es sich beim Programmieren um eine Tätigkeit handelt, die aufgrund eines intrinsischen Interesses betrieben wird und bei der eine Ausweitung des jeweiligen zeitlichen Rahmens wahrscheinlich ist.

---

20 Eine Beantwortung der Frage nach dem Stellenwert des Motivationsmittels ‚Geld' würde eine systematische Analyse der Beschäftigungsverhältnisse von bezahlten Entwicklern voraussetzen, die den Rahmen der vorliegenden Arbeit sprengt.

21 Gemeint ist hier das Engagement des Programmierers für die Entwicklung von freier Software generell.

Alternative Erklärungen und ihre Probleme

Nachdem wir im vorangegangenen Abschnitt die Gründe für eine Beteiligung herausgearbeitet haben, wollen wir uns mit den in der Literatur vertretenen Erklärungen des Phänomens beschäftigen. Einflußreich und richtungsweisend für die Erklärung der Beteiligung an freien Softwareentwicklungsprojekten hat das Working-Paper „The Simple Economics Of Open Source“ von Lerner/Tirole (2000)[22] gewirkt. Als erklärungsbedürftig wird hier die Frage präsentiert, weswegen Entwickler freiwillig zur Produktion eines öffentlichen Guts beitragen, wobei die Autoren eine Erklärung basierend auf altruistischen Motiven ausschließen (ebd. 2). Nach dieser Zuspitzung referieren die Autoren die Standardannahme der Kollektivgüterökonomie, nach der ein rationaler Akteur sich nur dann an der Bereitstellung eines Kollektivguts beteiligt, sofern das Kosten-Nutzen-Verhältnis positiv sei: „A programmer participates in a project whether commercial or open source, only if he derives a net benefit (broadly defined) from engaging in the activity. The net benefit is equal to the immediate payoff (current benefit minus current cost plus the delayed payoff.“ (ebd. 14, 2002: 212/213). Hieran anschließend nennen die Autoren folgende Faktoren, die für das Zustandekommen einer Beteiligung an freier Softwareentwicklung von Bedeutung sind: monetäre Kompensation, Nutzen einer Anpassung der Software an die eigenen Anwendungsbedürfnisse und Opportunitätskosten, wobei letztgenannte davon abhängig seien, wie angenehm die Tätigkeit empfunden wird. Zeitverzögert treten darüber hinaus als Nutzen die Anerkennung durch Peers sowie sogenannte ‚signaling incentives‘ auf, wobei hierunter die öffentliche Präsentation von Fähigkeiten gegenüber Peers, dem Arbeitsmarkt und potentiellen Kapitalgebern gemeint ist (2002: 214). Danach vergleichen die Autoren den Nutzen, der aus einer Beteiligung an freien Softwareentwicklungsprojekten resultiert, mit denen einer Beteiligung an proprietärer Softwareentwicklung. Die Vorteile freier Software liegen Lerner und Tirole folgend (a) in geringeren Kosten für die Einarbeitung in den Programmquellcode, da dieser durch seine Offenheit aus Schule oder Universität bereits bekannt sei, (b) in einem privaten Anwendungsnutzen durch die Möglichkeit einer Anpassung des Programms an eigene Bedürfnisse, (c) in der besseren Meßbarkeit der Leistungen durch Dritte und (d) in einer Motivationssteigerung durch

22 Eine leicht überarbeitete Version wurde mittlerweile veröffentlicht (vgl. Lerner/Tirole 2002). Das Vorgehen, der Aufbau und die Argumentation decken sich mit dem Working-Paper. Daneben findet sich eine kursorische Aufzählung des Nutzens einer Beteiligung an der freien Softwareentwicklung bei Lerner/Tirole 2001: 822 f.

ein höheres Maß an Eigenverantwortung. Weiter seien die erworbenen Programmierkompetenzen in einem geringeren Maße an die Produkte eines Unternehmens gebunden, so daß ein leichterer Wechsel des Arbeitsplatzes möglich sei (ebd. 216).

Die Struktur dieser Erklärung läßt sich nun wie folgt zusammenfassen: Die Autoren beobachten ein Verhalten (Beteiligung an freien Softwareentwicklungsprojekten), analysieren Kosten und Nutzen einer Beteiligung anhand modelltheoretischer Überlegungen (Opportunitätskosten, monetäre Kompensationen, Reputation usw.) und vergleichen das Verhalten mit einer aus ihrer Sicht naheliegenden Handlungsalternative (proprietäre Softwareentwicklung). Dabei läuft implizit die Annahme mit, freie Softwareentwickler verhielten sich nutzenmaximierend im Sinne einer Theorie rationaler Wahl. Im folgenden wollen wir uns mit den Problemen einer solchen Erklärung beschäftigen.

Ein erster, für uns näher zu betrachtender Punkt bildet die Annahme, einem Beitrag würde eine Wahlsituation vorausgehen, in der sich der Entwickler für eine Beteiligung entscheide. Hier bietet es sich an, zwei Fälle auseinanderzuhalten: zum einen den erstmaligen Beitrag, zum anderen Beiträge, denen bereits einige andere vorausgegangen sind und die in der Kontinuität eines anhaltenden Engagements stehen. Während im erstgenannten Fall die Annahme durchaus plausibel ist, daß es hier zu einer Entscheidungssituation kommt, in der der Akteur unterschiedliche Alternativen prüft, erscheint diese Annahme im Fall eines länger andauernden Engagements unrealistisch. Den Akteuren zu unterstellen, sie würden sich bereits bei der Erledigung von kleineren Beiträgen, wie der Meldung oder der Beseitigung eines einfachen Bugs, in einer Wahlsituation befinden, in der sie zumindest die Entscheidungsalternative einer Beteiligung gegenüber der einer Nicht-Beteiligung unter Berücksichtigung der sich aus der Wahl ergebenden Vor- und Nachteile abwägen, hieße nicht nur eine hohe Abwägungsbereitschaft zu unterstellen, sondern eine Ignoranz der Akteure gegenüber den durch Abwägungen fortlaufend anfallenden Transaktionskosten vorauszusetzen. Plausibler ist es statt dessen, in Fällen des fortgesetzten Engagements davon auszugehen, daß die Beteiligten Routinen entwickeln, die sie von der Notwendigkeit des fortgesetzten Entscheidens entlasten. In Anschluß an Mead (1973) wollen wir daher unterscheiden zwischen einem ungestörten Handlungsablauf, in dem der Handelnde selbst nicht in sinnvoller Weise von Reizen oder Reaktionen sprechen, sondern nur ein äußerer Beobachter diese Begriffe in den Handlungsablauf hineindeuten kann (vgl. Joas 1988: 424, 1989: 70) und einem unterbrochenen Handlungsablauf, bei dem ein Akteur über unterschiedliche Handlungsalternativen zu reflektieren beginnt. Hieran knüpfen wir die Vermutung, daß, je stärker sich die Betei-

ligung an einem Softwareprojekt dem Fall eines über einen längeren Zeitraum andauernden, kontinuierlich aus einer großen Anzahl kleiner Beiträge sich zusammensetzenden Engagements annähert, desto größer die Wahrscheinlichkeit einer Entwicklung von Routinen ist. Der erste Kritikpunkt an der oben dargestellten Erklärung lautet, daß Lerner/Tirole vorschnell Entscheidungssituationen unterstellen, wo Handeln auch anders als durch Entscheiden zustande kommen kann.

Die zweite Kritik, die wir gegen die von Lerner/Tirole entwickelte Erklärung vorbringen wollen, bezieht sich auf die Annahme, Entscheider betreiben eine Maximierung von Nutzen. Zutage tritt die Unterstellung in der Aufzählung des Faktors ‚Opportunitätskosten'. Der Nutzen einer gewählten Entscheidungsoption kann nur dann die Opportunitätskosten übersteigen, wenn die beste Entscheidungsoption gewählt wird. Bereits bei der Wahl der zweitbesten Alternative wäre der entgangene Nutzen höher als der durch die Wahl realisierte. Bei Lerner/Tirole bleibt es nun offen, ob unter Opportunitätskosten sämtliche andere denkbare Entscheidungsalternativen behandelt werden, auf die der Akteur seine Zeit hätte verwenden können, oder ob damit lediglich die Abwägung gegenüber der im folgenden dann ausführlicher diskutierten Alternative einer proprietären Softwareentwicklung gemeint ist. Gegen die erste Lesart, mit Opportunitätskosten sei der entgangene Nutzen nicht weiter spezifizierter Entscheidungsalternativen gemeint, lassen sich einige mittlerweile als klassisch zu bezeichnende Einwände vorbringen. Hier sei der von March und Simon formulierte Zweifel daran, Entscheider könnten unter den Bedingungen der Knappheit von Zeit sowie unsicherer und unvollständiger Informationen sämtliche Entscheidungsalternativen berücksichtigen und bewerten, angeführt (March/Simon 1977; March 1990). Realistischer erscheint die von den Autoren vertretene These, Entscheider betreiben eine Strategie des *Satisficing*, wählten also die erstbeste Option, die ein bestimmtes Anspruchsniveau befriedigt (Simon 1957: 204, 253 f.).[23] Problematisch an der zweiten Lesart ist demgegenüber, daß Lerner/Tirole zwei Alternativen formulieren, von denen sie annehmen, sie würden sich den Entscheidern stellen, ohne allerdings die Richtigkeit dieser Unterstellung empirisch zu überprüfen oder Argumente vorzubringen, weswegen sich die Alternativen gerade auf diese beiden Optionen beschränken. Wie das Zitat aus dem ‚Inital Announcement' zu Beginn des Kapitels zeigt, kann die Handlungsalternative für manche Programmierer auch lauten: Entwicklung von freier Software oder Verzicht auf eine weitere Beschäftigung mit dem Computer. Zudem ist die

23 Das Anspruchsniveau wird dabei in Fällen, in denen es schnell oder gar nicht erreicht wird, nach oben oder nach unten angepaßt.

Handlungsalternative einer proprietären Softwareentwicklung für die weit überwiegende Zahl der an freien Softwareentwicklungsprojekten Beteiligten gar nicht verfügbar.[24] Der Punkt sollte deutlich geworden sein: Problematisch an der zweiten Lesart ist, daß der Rational-Choice Theoretiker den Akteuren durch die Annahme von objektiv gegebenen Handlungsalternativen Alternativen unterstellt, die sich für diese auch anders darstellen oder aufgrund von hohen Anforderungen nicht gewählt werden können. Zugespitzt formuliert besteht das Risiko der Formulierung von Entscheidungsalternativen, ohne die den Möglichkeitsraum beschränkenden Mechanismen angeben zu können, schlicht darin, von falschen Alternativen auszugehen.

Neben der prinzipiellen Skepsis gegenüber der Möglichkeit einer Nutzenmaximierung und der Gefahr, Entscheidern falsche Handlungsalternativen zu unterstellen, ist die Struktur der Erklärung selbst problematisch. Die Autoren beobachten das Verhalten von Akteuren (Beteiligung an freien Softwareentwicklungsprojekten) und schließen von diesem auf die Präferenzen, von denen sie unterstellen, sie bilden die Ursachen für das Verhalten. Diese Erklärungsfigur ist in der Literatur unter der Bezeichnung ‚revealed preferences' bekannt.[25] Ebenso wie unter Punkt 2 besteht hier ein Irrtumsrisiko einer fälschlichen Zurechnung, wenn nämlich ein Verhalten nicht nur durch die unterstellte Ursache, sondern auch durch andere Faktoren zustande gekommen sein kann.[26] Neben den unterstellten Präferenzen kommen als Verursacher eines Verhaltens vor allen Dingen normative Überzeugungen infrage. Auch hier sei wiederum nur knapp auf das dem Kapitel voranstehende Zitat von Stallman verwiesen: Ein Kandidat für eine normative Verursachung findet sich hier im emanzipatorischen Ziel der Vermeidung von Abhängigkeitsbeziehungen von Softwareunternehmen und der Etablierung einer Gabenökonomie, die das Prinzip einer wechselseitigen Hilfeleistung mit beinhaltet. Aufgrund der Möglichkeit einer Verursachung des Verhaltens ‚Be-

24 Dies gilt nicht nur für Entwickler, die sich mit kleinen Beiträgen, wie der Implementation einer einzelnen Funktionen oder der Beseitigung einfacher Bugs, beschäftigen, sondern selbst für hoch engagierte Entwickler. Sofern diese im Alleingang ein Programm entwickeln, können sie es durchaus mit einer proprietären Lizenz schützen. Um dann aber monetäre Gewinne zu realisieren, müßten Vertriebswege aufgebaut und das Programm beworben werden. Dies bedeutet allerdings Aufwand, der sich nur in wenigen Fällen lohnen und zudem viele Entwickler überfordern dürfte.

25 Dies muß zumindest Lerner/Tirole unterstellt werden, da sie die Präferenzen von freien Softwareentwicklern nicht direkt erheben, sondern durch die Interpretation des Verhaltens zu erschließen suchen.

26 Zu der Kritik an Erklärungen vom Typ ‚revealed preferences' siehe auch ausführlicher Sen 1986 insbes. S. 71 f.

teiligung an freien Softwareprojekten‘ durch mehr als einen Faktor (normative Zielsetzungen oder individuelle Präferenzen) verbietet sich eine Erklärung vom Typ ‚revealed preferences‘.

Während sich der dritte Kritikpunkt auf die logische Struktur der Erklärung bezieht, wollen wir uns nun den einzelnen Handlungsfolgen zuwenden, von denen Lerner/Tirole unterstellen, sie würden von den Akteuren angestrebt werden. Unter diesen Handlungsfolgen finden sich auch solche, bei denen fraglich ist, ob die intentionale Verfolgung überhaupt möglich ist. Handlungsfolgen dieser Art diskutiert Elster unter dem Stichwort „Zustände, die wesentlich Nebenprodukt“ sind (Elster 1986: 143 ff.) und meint damit solche, die möglicherweise erwünscht sein können (es aber nicht müssen), jedoch von einem Akteur nicht direkt angestrebt werden können. Beispiele hierfür bilden ‚Natürlichsein‘, Schlaf, Vergessen oder Glauben (im religiösen Sinne des Wortes). Als intellektuellen Fehlschluß bezeichnet Elster nun den Versuch, einen Zustand, der wesentlich Nebenprodukt ist, „als Ergebnis einer Handlung zu erklären, die zu seiner Herbeiführung entworfen wurde“ (ebd. 141). Einen solchen Fehlschluß begehen Lerner/Tirole im Fall der Reputation: Wie weiter oben ausgeführt, handelt es sich dabei um einen Ruf, der auf vorangegangene Leistungen basiert, die von anderen Personen erinnert und der betreffenden Person zugerechnet werden.[27] Problematisch an der These, eine Beteiligung an freien Softwareentwicklungsprojekten fände zum Zweck des Reputationserwerbs statt, ist, daß es zum Zeitpunkt, an dem ein Beitrag erbracht wird, nicht absehbar ist, ob die Leistung von anderen Personen wahrgenommen wird und sich damit das erzielte Verhältnis von Aufwand und (Reputations-)Ertrag der Kontrolle des Akteurs entzieht. Erschwerend kommt als Eigenschaft von Reputationssystemen hinzu, daß die Beteiligten jeden Anschein vermeiden müssen, sie strebten Reputation an, so daß die Möglichkeit einer nachträglichen Reklamierung eines nicht angemessenen Maßes an Reputation ebenso ausfällt wie eine Strategie, auf die erbrachten Leistungen aufmerksam zu machen.[28] Diese Eigenschaft von Reputation machen es daher unwahrscheinlich, daß sich Akteure vorrangig zum Zweck deren Erwerbs an der

27 Vgl. hierzu auch Raymonds Erläuterungen zu Reputation in „How to become a hacker“: „Accordingly, when you play the hacker game, you learn to keep score primarily by what other hackers think of your skill (this is why you aren’t really a hacker until other hackers consistently call you one).“ (1999: 242) In diesem Zitat macht der Autor deutlich, daß der Ruf einer Person weder von der eigenen Einschätzung noch von einer ‚objektiven‘ Leistung abhängig ist, sondern von deren Beurteilung durch *andere* Hacker.

28 Ähnlich kritisch zur motivationalen Rolle von Reputation äußern sich auch Osterloh/Rota/Kuster 2002.

Entwicklung von freier Software beteiligen, sondern daß es sich hierbei um eine positive bewertete, aber eben nicht (primär) angestrebte Nebenfolge handelt.

Ein ähnliches Problem der vorschnellen und ‚theoriegesteuerte Dateninterpretation' (Green/Shapiro 1999: 58) findet sich in bezug auf den Faktor ‚aufgewendete Zeit'. Weiter oben hatten wir das intrinsische Interesse, mit der die Entwickler der Programmiertätigkeit nachgehen, herausgearbeitet. Von Lerner/Tirole wird Zeit konsistent im Bezugsrahmen ihrer Theorie als Kostenfaktor interpretiert, da während der für die Entwicklung aufgewendeten Zeit keine anderen Tätigkeiten durchgeführt werden können. Vor dem Hintergrund des intrinsischen Interesses stellt sich nun die Frage, ob die Zeit von den Akteuren in der Kategorie ‚Kosten' wahrgenommen, oder ob sie nicht eher ‚verbracht' wird, ohne daß sie bei Entscheidungskalkülen berücksichtigt und ohne daß ihr Verstreichen groß zur Kenntnis genommen wird. Hierfür spricht zumindest die Untersuchung von Lakhani/Wolf (2003: 11), in der von den befragten 684 Programmierern 61 Prozent die Beteiligung als die kreativste Tätigkeit ansehen, die sie bislang ausgeübt hätten und sogar 73 Prozent angaben, sie würden während ihrer Entwicklungstätigkeit ‚immer' oder ‚häufig' das Zeitgefühl verlieren. Akteure, denen die Dimension Zeit als Kostenfaktor gilt, wären mit einem solchen Verhalten wenig kostenbewußt. Näher scheint daher die Interpretation zu liegen, die Dimension Zeit bildet hier keinen (prominenten) Faktor in einem Kosten/Nutzen-Kalkül.[29]

Zusammenfassend ist die von Lerner/Tirole angebotene Erklärung unbefriedigend. Erstens muß ein Erklärungsmodell des Zustandekommens von Beiträgen zu freier Softwareentwicklung neben Entscheidungs- oder Wahlsituationen auch den Normalfall einer sich kontinuierenden Beteiligung berücksichtigen. Hier ist dann nicht mit Alternativenabwägung, sondern mit Routinen zu rechnen, wobei sich dann allerdings die Frage nach deren Herkunft stellt. Zweitens erscheint die Möglichkeit einer Nutzenmaximierung bei einer großen Anzahl von Entscheidungsalternativen nicht praktikabel, während bei einer Einschränkung des Alternativenspektrums, innerhalb dessen ein Akteur wählt, diejenigen Mechanismen benannt werden müßten, die für eine Limitation des Alternativenspektrums verantwortlich sind. Drittens ist eine Erklärung mit ‚revealed preference'-Struktur nur dann möglich, wenn ausgeschlossen werden kann, daß das Verhalten nicht auch durch andere Faktoren zustande gekommen ist. Dies ist hier nun allerdings gerade nicht

29 Vgl. auch Zeitlyn 2003. Dieser hebt als Merkmal von nicht-marktförmigen Systemen hervor, daß Zeit hier nicht gemessen und von den Akteuren nicht in Kosten-Nutzen-Kalkulationen einbezogen wird (ebd. 1287).

der Fall. Viertens unterstellen die Autoren den Akteuren, sie würden sämtliche positive Nebenfolgen intentional verfolgen, wobei Lerner/ Tirole auch Handlungsfolgen wie Reputation nennen, bei der eine intentionale Verfolgung schwer vorstellbar ist. Fünftens unterstellen sie den Programmierern, sie würden Faktoren ebenso wie Theorien rationalen Wahlverhaltens in den Kategorien von Kosten und Nutzen interpretieren. Hier hatten wir am Beispiel des Faktors ‚Zeit' begründeten Zweifel angemeldet. Am problematischsten ist jedoch die Unvollständigkeit der Erklärung. Weder über die Mechanismen, die für eine Einschränkung der Handlungsalternativen sorgen, noch über die Herkunft der Präferenzen werden hier Aussagen getroffen.

Eine komplexere, der motivationalen Psychologie entstammende Erklärung findet sich bei Hertel/Niedner/Herrmann 2003. Die Autoren kombinieren ein Faktorenmodell, das ursprünglich zur Erklärung einer Beteiligung an sozialen Bewegungen entwickelt wurde, mit einem motivationalen Modell von Kleingruppen. Beide Modelle lehnen sich an das in der Sozialpsychologie verbreitete Modell von Höhe des Werts eines Ziels und der Wahrscheinlichkeit seines Eintretens an (Erwartung mal Wert-Konstrukt). Das erste Modell stellt mit den folgenden Faktoren eine Erweiterung des ‚Erwartung mal Wert'-Konstrukts dar: (a) ‚Collective motives', definiert als Bewertung des Ziels multipliziert mit der Wahrscheinlichkeit der Zielerreichung. (b) ‚Social motives/norm-oriented motives', definiert als die Bewertung des Verhaltens durch signifikante andere. (c) ‚Reward motives', definiert als die anfallenden Kosten und Nutzen. (d) ‚Identification process', definiert als die Zugehörigkeit zu der ein Handlungsziel verfolgenden Gruppe (ebd. 1163). Demgegenüber konkretisiert das motivationale Modell der Kleingruppen die Faktoren ‚Erwartung' und ‚Wert'. Neben dem Wert (hier als ‚valence' bezeichnet) wird der Faktor ‚Erwartung' aufgeschlüsselt nach der Relevanz des eigenen Beitrags zur Erreichung des Gruppenziels (*instrumentality*), die Einschätzung eigener Fähigkeiten, den Beitrag leisten zu können (*self-efficacy*), sowie das Vertrauen darauf, das andere Teammitglieder ebenfalls Leistungen erbringen und die Beiträge anderer nicht ausnutzen (*trust*) (dies. 1164).

Im Unterschied zu Lerner/Tirole vermeiden die Autoren die problematische und kaum durchzuhaltende Annahme einer individuellen Nutzenmaximierung durch die Wahl der Handlungsoption ‚Projektbeteiligung'. An diese Stelle tritt die probabilistische Annahme, eine Beteiligung sei um so wahrscheinlicher, je höher der Wert und die Chancen der Zielerreichung von den Akteuren eingeschätzt werden und je günstiger das Kosten/Nutzen-Verhältnis sei. Trotz guter empirischer Bestätigung bleibt unbefriedigend, daß auch hier wiederum verschieden Faktoren in

den Kategorien von Kosten und Nutzen behandelt werden – was, wie bereits weiter oben hervorgehoben wurde, zumindest für den Faktor Zeit problematisch erscheint. Ebenfalls kritisch hervorzuheben ist die Unvollständigkeit des Erklärungsmodells. Die Faktoren ‚Bewertung des Handlungsziels' und ‚Bewertung von Verhaltensweisen durch signifikante Andere' werden als Explanandum geführt und scheinen den Autoren zufolge nicht erklärungsbedürftig zu sein.

Ein dritter Typ von Erklärung einer Beteiligung an freien Softwareentwicklungsprojekten findet sich bei Osterloh/Rota/Kuster o.J. Diese entwickeln auf der Basis der Motive für eine Beteiligung eine Idealtypologie, die folgende Akteure umfaßt: ‚Commercial Service Provider', die monetäre Profite anstreben; ‚Lead User', die mit der Entwicklung eigene Anwendungsbedürfnisse befriedigen; ‚Reputation Investors', die durch die Tätigkeit anderen Personen ihre Fähigkeiten signalisieren und damit einen monetären Nutzen anstreben; ‚Fun Seekers', die sich aus Interesse an der Tätigkeit an der Entwicklung beteiligten, sowie ‚Members of the Tribe', die sich vorrangig aufgrund einer generalisierten Reziprozitätsnorm zur Entwicklung veranlaßt sehen.

Gegen diese Typologisierung läßt sich nun mit Blick auf die unserer Untersuchung zugrunde liegenden Interviews einwenden, daß die Entwickler jeweils mehrere Motive für eine Beteiligung am Projekt nennen – ein Merkmal, das für sämtliche Interviews gilt. Das Charakteristikum der Kombination von mehreren Motiven wird durch eine solche Typologie unterschlagen. Hinzu kommt, daß die Autoren der Möglichkeit des direkten Anstrebens von Reputation und der Transformation dieser in monetäre Profite oder in die Verbesserung von Karrierechancen sehr kritisch gegenüberstehen, ‚Reputation Investor' dennoch als eigenständiger Typ geführt wird.

Zusammenfassung

Nachdem wir in diesem Kapitel anhand der Interviews die Gründe herausgearbeitet haben, die für das Zustandekommen von Beiträgen zur Entwicklung von freier Software verantwortlich sind, und die in der Literatur vorzufindenden Erklärungen kritisch durchgesehen haben, wollen wir diese Erträge zu einem Modell verdichten. In der Auseinandersetzung mit den in der Literatur vertretenen Erklärungen haben wir gesehen, daß sowohl eine Typologisierung der Entwickler auf der Basis der ihrer Beteiligung zugrunde liegenden Motivationen als auch Rational-Choice-Erklärungen Schwächen besitzen: Gegen die Typologisierung der Entwickler sprechen die empirischen Befunde unserer Untersu-

chung, daß für das Zustandekommen einer Beteiligung jeweils mehrere Motive verantwortlich sind, gegen Rational-Choice-Theorien die weiter oben ausgearbeiteten Kritikpunkte. Um unsere Ergebnisse zusammenzufassen, schlagen wir vor, zwischen drei Typen von Faktoren zu unterscheiden, die eine Beteiligung an freier Softwareentwicklung begünstigen: *(a) Individuelle Motivationen*: Hierzu zählt das intrinsische Interesse an der Programmiertätigkeit, aber auch ein persönlicher Nutzen, wie die Befriedigung von eigenen Anwendungsbedürfnissen zur Implementation gewünschter Funktionalität. *(b) Rahmenbedingungen freier Softwareentwicklung*: Hierunter fallen die Faktoren, die auf freie Softwarelizenzen (z.B. Verpflichtung zur Veröffentlichung von Modifikationen im Fall einer Anpassungen der Software an eigene Anwendungsbedürfnisse) oder strukturelle Bedingungen offener Entwicklungsprojekte zurückzuführen sind, und *(c) institutionalisierte Orientierungen*, wie die normative Verpflichtung zur Gegengabe oder emanzipatorische Ziele wie die Vermeidung von Abhängigkeiten.

Entscheidend ist nun, daß diese Faktoren ein komplexes Bündel bilden und die Kombination dieser sich von Entwickler zu Entwickler unterscheiden kann. Die Spannweite der Gründe für eine Beteiligung reicht dabei von Entwicklern, deren Beteiligung stark von persönlichen Kosten-/Nutzen-Kalkülen geprägt ist, bis hin zu Programmierern, deren Verhalten weitgehend von institutionalisierten Normen bestimmt ist. Ein Beispiel für die erste Extremposition bilden sporadische Beteiligte, die ein Programm ihren Bedürfnissen anpassen und aufgrund von freien Softwarelizenzen dazu verpflichtet sind, ihre Modifikationen dem Softwareentwicklungsprojekt zugänglich zu machen.[30] Die Entwicklungstätigkeit resultiert hier ausschließlich aus egoistischen Anwendungsinteressen, die Weitergabe der modifizierten Version aus dem ‚constraint' freier Softwarelizenzen und nicht aus einer Orientierung an Normen. Die paradigmatischen Beispiele für eine ausschließlich normenregulierte Beteiligung an freier Softwareentwicklung bilden Personen, die emanzipatorische Ziele verfolgen oder von der Verpflichtung zur Gegengabe geleitet sind.[31] In den meisten Interviews zeigen sich jedoch unterschiedliche Gründe, die weder ausschließlich auf rationales Kalkül noch ausschließlich auf normenreguliertes Handeln hinweisen. Dazu läßt sich beobachten, daß sich die Gründe für ein Engagement in Abhängigkeit

30 Auf den Mailinglisten von freien Softwareprojekten zeichnen sich diese Beteiligten dadurch aus, daß sie ein oder wenige Male Beiträge an die Mailingliste schicken, mit denen sie ihre modifizierten Versionen an das Projekt übermitteln.

31 Als paradigmatisch hierfür kann erneut das Eingangszitat dieses Kapitels von Richard M. Stallman gelten.

von Dauer und Intensität verändern. Im folgenden wollen wir einige Plausibilitätsüberlegungen zu den einzelnen Gründen anstellen und damit vorsichtige Tendenzaussagen zum Stellenwert der einzelnen Faktoren für die Erklärung des Zustandekommens von Beteiligung treffen.[32]

Typisch für die Beteiligten ist das intrinsische Interesse an der Programmiertätigkeit selbst. Dies gilt für sämtliche interviewte Entwickler; Ausnahmen könnten hier möglicherweise Personen bilden, die von Dritten mit der Implementation einer bestimmten Funktionalität beauftragt und dafür bezahlt werden, nicht aber für das Gros der Entwickler, das dieser Tätigkeit in seiner Freizeit nachgeht. Lob von Anwendern und die Befriedigung eigener Anwendungsbedürfnisse stellen Gründe dar, von denen plausibel angenommen werden kann, daß sie zu Beginn des Engagements eine wichtige Rolle spielen, danach jedoch weniger relevant werden. Nach der Implementation der gewünschten Funktionalität entspricht ein Programm den eigenen Anwendungsbedürfnissen, das Lob von Anwendern verschleißt, da es durch Wiederholung zur Selbstverständlichkeit wird. Hinsichtlich Reputation ist es wenig wahrscheinlich, daß sie direkt angestrebt wird und bereits bei der Aufnahme der Entwicklungstätigkeit eine Rolle spielt. Allerdings ist zu vermuten, daß sie in fortgeschrittenen Stadien der Beteiligung einen stabilisierenden Faktor für das Engagement darstellt. Ebenfalls zu einem späteren Zeitpunkt des Engagements dürften emanzipatorische Ziele die Beteiligung normativ regulieren: In den Interviews zeigt sich, daß eine Beschäftigung mit dem normativen Deutungshorizont, wie den gesellschaftspolitischen Texten von Stallman, in der Regel erst im Zuge eines bereits länger anhaltenden Engagements stattfindet, Wirkungen dieses Faktors also erst eine Auseinandersetzung mit den für einen Teil der freien Softwareentwickler zentralen normativen Leitvorstellung voraussetzt. Hier ist allerdings zu bemerken: Die Beschäftigung mit diesen Ideen muß nicht zwangsläufig dazu führen, daß die Entwicklungstätigkeit von normativen Gründen motiviert wird, sondern kann, wie zumindest ein Interview zeigt, auch zu Indifferenz gegenüber den Normen und dem dort entwikkelten Orientierungshorizont führen. Die folgende Tabelle faßt unsere Tendenzaussagen zum Stellenwert der Gründe für eine Beteiligung in Abhängigkeit zur Intensität des Engagements summarisch zusammen.

32 Diese Aussagen wollen wir als Hypothesen verstanden wissen, die selbst einer empirischen Überprüfung bedürfen. Dies ist allerdings nicht im Rahmen dieser Untersuchung, sondern nur durch Längsschnittstudien zu leisten, in denen die Beteiligten zu verschiedenen Zeitpunkten nach ihren Motiven befragt werden.

Tabelle 2: Gründe für eine Beteiligung an Softwareentwicklungsprojekten

Motiv	Aufnahme des Engagements	Kontinuierliches Engagement	Hohes Engagement
Individuelle Anreize			
Intrinsisches Interesse	—	—	→
Eigene Anwendungsbedürfnisse	—	→	
Monetäre Anreize		—	→
Situative Gründe			
Realisierung weitreichender Entwicklungsziele	—	—	→
Größere Verbreitung des Programms	—	—	→
Hohe Entwicklungs-geschwindigkeit, keine alleinige Verantwortung	—	—	→
Anerkennung von Anwendern	—	→	
Institutionelle Gründe			
Reputation		—	→
Emanzipatorische Ziele		—	→

Projektbeginn: Widersprüchliche Anforderungen

In der ex post-Betrachtung scheint es eine Selbstverständlichkeit zu sein, daß erfolgreiche Projekte irgendwann einmal gegründet wurden: Auf den Apache-Webserver hat der Markt scheinbar nur gewartet, Linux entstand zu einem Zeitpunkt, zu dem es zum Windows-Betriebssystem kaum eine Alternative gab, KDE und Gnome wurden ins Leben gerufen, als grafische Benutzeroberflächen visuell immer ausgefeilter wurden. Das Rezept für freie Softwareentwicklungsprojekte scheint schlicht zu lauten: Die richtige Idee zum richtigen Zeitpunkt zu haben (oder vielleicht auch: schnell genug auf fahrende Züge aufzuspringen), führt zu einem erfolgreichen Projekt. In diesem Kapitel werden wir die Frage nach dem Ereignis der Projektgründung zuspitzen, indem wir die Vorstellung, sie sei jederzeit erfolgreich möglich, hinterfragen und die Problemfassung erweitern: Eine Projektgründung erschöpft sich nicht darin, zu einem glücklichen Zeitpunkt eine gute Idee zu haben, vielmehr geht es sachlich um eine zumindest vorläufige Bestimmung eines Entwicklungsziels, sozial um die Integration von Entwicklern und den Aufbau einer Struktur, die ein arbeitsteiliges Vorgehen in Richtung eines gemeinsamen Projektziels ermöglicht.

Da wir im Fall von KMail keinen Zugang zum Material haben, das über die Anfänge des Projekts Auskunft gibt – das Archiv der Mailingliste reicht nicht bis zur Projektgründung zurück,[1] die interviewten Pro-

1 Nach Auskunft eines Interviewpartners wurde KMail Anfang des Jahres 1997 gegründet, die Mailingliste erst später eingerichtet. So findet sich im Mailinglistenarchiv nur ein Monat des Jahrgangs 1997, während der Jahr-

grammierer sind selbst keine Gründungsmitglieder – werden wir die ‚Stunde Null' offener Programmierprojekte anhand eines Interviews mit einem Entwickler analysieren, der über vielfältige Erfahrungen verfügt. Für unsere Frage nach den Bedingungen einer Projektgründung besonders einschlägig ist ein Versuch, der allerdings fehlschlägt. Anhand der Rekonstruktion des Scheiterns der Projektgründung läßt sich gut erkennen, welche Aufgaben hier gelöst werden müssen. Wir werden sehen, daß sich die Anforderungen nach einem kohärenten Konzept und der Offenheit gegenüber anderen Entwicklungszielen widersprechen.

Anschließend gehen wir der Frage nach, wie die Entwickler diese Anforderungen bewältigen. Hier werden wir eine häufig eingesetzte Technik – die Modularisierung des Programmquellcodes – vorstellen und zeigen, daß diese ein Mittel zur Entschärfung der Widersprüchlichkeit bildet. Bei der Modularisierung handelt es sich nun allerdings nicht um eine Strukturierungstechnik, die ihre Ursprünge im Bereich der freien Softwareentwicklung hat. Mit Rückgriff auf das Software Engineering werden wir darstellen, daß die Idee der Zerlegung von komplexen Aufgaben in überschaubare Teilprobleme bei eindeutiger Definition der Schnittstellen bereits länger diskutiert wird und in der freien Softwareentwicklung eine neue Kontextualisierung erfährt.

In einem dritten Schritt werden wir diese Befunde in die Technikgeneseforschung einordnen. Das hier zu entwickelnde Argument lautet, daß die Gestalt der Technik ihren Ursprung nicht in den Interessen der beteiligten Akteure, Orientierungshorizonten oder Systemlogiken hat – also den Faktoren, durch die die Technikgeneseforschung gemeinhin die soziale Gestalt von Technik erklärt –, sondern aus den praktischen Anforderungen des Entwicklungsprozesses selbst resultiert. Für die Technikgeneseforschung wird damit die Frage nach den Spuren aufgeworfen, die die Praxis der Technikentwicklung am Artefakt hinterläßt.

Das Scheitern einer Projektinitiierung

Wie hat man sich die Gründung eines freien Softwareentwicklungsprojekts vorzustellen, wie wird es ins Leben gerufen und welche Aufgaben müssen der oder die Gründer dabei bewältigen? Im folgenden wollen wir einen Gründungsversuch eines relativ schnell scheiternden Projekts analysieren, um einen Zugriff auf die Beantwortung dieser Frage zu er-

gang 1998 völlig fehlt und der Jahrgang 1999 nur zwei Monate umfaßt. Vollständig ist das Archiv erst ab dem 01.01.2000.

halten. Der Interviewpartner, der die Gründung selbst maßgeblich vorangetrieben hat, schildert die Projektinitiierung wie folgt:

„Das kürzlich, das war ein Kryptoprojekt, wo wir halt so ’nen PGP-Programm basteln wollten in Java. Da war ein relativ hoher Koordinationsaufwand von Nöten. Da gabs auch gleich von Anfang an mehrere Entwickler. Und im Endeffekt, das war da so das Problem, daß das Programm sich nicht sauber in Teile teilen ließ. Also man konnte nicht irgendwie sagen, ‚ich mach dies, ich mach jenes‘, sondern es gab irgendwie ein CVS-Repository und man kam sich öfter in die Quere und machte irgendwie Änderungen die miteinander in Konflikt standen und es gab auch relativ viel Design. Also ich hatte den Anspruch, ich wollte das nicht alleine machen, sondern es sollten eben andere Leute da mitarbeiten und beim Design mit dabei sein und es gab ’ne Libary, die schon existierte, da hab ich mit dem Entwickler relativ viel kommuniziert über das Design, weil ich das halt viel zu kompliziert fand und auch ein paar Features fehlten die dann nicht so einfach zu integrieren waren. […] Wir hatten da ’ne unterschiedliche Ansicht in der Philosophie. Und das hat einen tierischen Kommunikationsaufwand nach sich gezogen, der im Endeffekt nichts gebracht hat. […] Also, er hat mich nicht von seiner Meinung überzeugen können und ich ihn nicht von meiner.“ (D3)

Auffällig an dieser Passage ist bereits die Selbstverständlichkeit, mit der der Interviewte den Vorgang der Projektgründung schildert, die insbesondere durch die Formulierung „halt so ’nen PGP-Programm basteln“ zum Ausdruck kommt. Eine Projektgründung ist offensichtlich eine weitgehend undramatische Begebenheit ohne herausgehobenen Ereignischarakter.[2] Weiter scheint zu Beginn des Projekts die Erwartung an dessen Erfolg nicht allzu groß zu sein, so daß ein Scheitern ebensowenig dramatisch ist wie die Gründung.[3] Im obigen Zitat zählt der Interviewte einige Rahmenbedingungen auf, die den Versuch kennzeichnen: Es finden sich Entwickler zusammen, die gemeinsam an der Entwicklung des

2 Diese Selbstverständlichkeit erklärt zumindest zum Teil die große Anzahl von freien Softwareprojekten. Die Anzahl schwankt täglich und ist schwer zu bestimmen. Allein auf der Plattform Sourceforge werden 82.950 Projekte gehostet, vgl. http://sourceforge.net/index.php. Vermutlich befinden sich nicht alle in einem Stadium aktiver Entwicklung. Die Untersuchung von Ghosh/Robles/Glott 2002 basiert aber immerhin auf einem Sample von 16.485 Projekten.

3 Einer alternativen Lesart, daß es sich bei dieser Schilderung um eine nachträgliche Uminterpretation im Zuge einer Enttäuschung handelt, gehen wir nicht weiter nach, da wir in der Darstellung des Interviewten keine Hinweise für eine Erwartungsenttäuschung finden. Im Gegenteil setzt sich der Entwickler nach wie vor dem Risiko des Scheiterns aus, indem er sich weiterhin an Projekten beteiligt und Gründungsversuche unternimmt.

Programms arbeiten wollen, das Programm soll auf dem PGP-Verfahren[4] aufbauen, und – wie im Verlauf des Interviews noch deutlich wird – zur Verschlüsselung privater E-Mails genutzt werden. Die Programmiersprache Java, in der das Programm implementiert werden soll, steht ebenso fest wie die Nutzung eines Concurrent Version System (CVS) zur Verwaltung der Entwicklungsversionen und zur Vermeidung von Konflikten zwischen parallel stattfindenden Veränderungen. Zu diesem Zeitpunkt existieren also ein recht allgemein gehaltenes Projektziel, eine Gruppe von Personen, die das Ziel verfolgen möchte sowie recht allgemeine Vorstellungen über die Organisation des Projekts. Neben diesen Fixpunkten kreist die Beschreibung des Entwicklers um einen Aspekt, den er für das Scheitern des Projekts verantwortlich macht: Es ist die Frage, ob das Projekt auf einer bereits existierenden Programmbibliothek aufbauen oder ob die Bibliothek neu geschrieben werden soll. Eine Einigung auf ein gemeinsames Vorgehen bleibt jedoch aus und die Entwicklungsarbeit wird trotz fehlender Einigung mit der Implementation einer PGP-Programmbibliothek aufgenommen.

Die hier vom Interviewten thematisierten Schwierigkeiten verweisen auf die erste zum Projektbeginn zu lösenden Aufgabe: Es ist notwendig, daß neben der Festlegung eines allgemeinen Entwicklungsziels auch eine möglichst weitgehende Verständigung auf ein gemeinsames Vorgehen im Projekt stattfindet. Schwierig kann es werden, wenn sich an diesem Klärungs- bzw. Definitionsprozeß mehrere Personen beteiligen: Hier kann es dann – wie in unserem Beispiel – zu der Situation kommen, daß von den Beteiligten unterschiedliche Vorgehensweisen präferiert werden. Der Anspruch des Interviewten, die grundlegenden Designentscheidungen nicht allein treffen zu wollen, bildet Ausdruck eines Dilemmas. Die Entscheidung über den Programmentwurf kann zwar von einer einzelnen Person getroffen werden,[5] hat aber möglicherweise zufolge, daß andere Entwickler nicht dazu bereit sind, an der Realisierung der vorgegebenen Entwicklungsziele mitzuarbeiten, während die frühzeitige Beteiligung weiterer Entwickler zu Problemen bei der Entscheidungsfindung und dem Ausbleiben einer Einigung über Ziel und Vorgehen führen kann. Das Risiko des Entstehens eines Ein-Personen-Projekts kann also nur durch die Inkaufnahme eines Dissens-Risikos bezüglich

4 PGP bildet ein Akronym für Pretty Good Privacy, einem Verschlüsselungsverfahren, das sich nicht nur unter den Entwicklern großer Beliebtheit erfreut. Eine Implementation des Verfahrens vertreibt die pgp Corporation (siehe http://www.pgp.com). Da es sich hierbei nicht um freie Software handelt, wurde innerhalb des GNU-Projekts eine freie Implementation des Verfahrens entwickelt (http://www.gnupg.org).

5 Beispiele bilden Linux, Perl und Sendmail (Lerner/Tirole 2002: 208 ff.).

der Zielsetzungen und dem Vorgehen vermieden werden. Darin erschöpft sich nun aber nicht die Aufgabe einer Projektgründung, wie die weitere Schilderung zeigt.

„Das Grundproblem war eigentlich, daß ich in dieser Libary relativ häufig das Design geändert habe. Und dann änderten sich die externen Schnittstellen und dieses Grafikteil, diese graphische Schnittstelle benutzte der [andere Entwickler, N.C.T.] ja. Und dann wars halt so, an der einen Seite war Frank, das war der andere Entwickler, der machte dann halt ein paar Fixes – und ich hatte inzwischen ein paar Klassen komplett umgeschmissen, und dann gabs halt ein Konflikt beim Mergen.[6] Das war einfach ziemlich frustrierend. Weil dann konnte er wieder nicht weiterarbeiten. Also, das lag jetzt weniger an CVS an sich, das da irgendwelche Fehler gewesen wären, als daß das Design eines Teiles sich sehr schnell änderte und ein anderer Teil versuchte, schon darauf aufzubauen. Und rannte dem ständig hinterher. Und das klappte eben nicht. Das konnte auch eigentlich nicht klappen, also das war nicht 'nen Problem des Werkzeugs, sondern wir haben einfach auf 'ne Art und Weise programmiert, die nicht funktionieren konnte.“ (D 3)

Das Zitat verdeutlicht eindrücklich, daß eine Projektgründung auch die Entwicklung und Etablierung einer Form von Arbeitsteilung beinhaltet. Die in unserem Fall gefundene Arbeitsteilung – der Interviewte entwickelt die Programmbibliothek, während sich der andere Programmierer um die grafische Benutzerschnittstelle kümmert – funktioniert nicht, weil eine gleichzeitige Bearbeitung von Aufgaben versucht wird, die nur nacheinander sinnvoll erledigt werden können. Das komplexe Problem der Programmentwicklung muß also derart aufgeteilt werden, daß eine Kooperation überhaupt erst möglich wird. Abstrakt formuliert geht es dabei um die Erhöhung von Komplexität in der Zeitdimension (durch die Gleichzeitigkeit der Arbeit) bei Verringerung der sachlichen Komplexität (der einzelnen Teilaufgaben).

Damit jedoch nicht genug. In der Schilderung des Interviewten tauchen weitere Entwickler auf, die sich für das Projekt interessieren und beteiligungswillig sind, sich aber im Zuge des Ausbleibens einer Einigung auf einen grundlegenden Programmentwurf vom Projekt abwenden. „Es[7] führte eher dazu, das halt Leute, die da einen Bezug hatten, auf der Mailingsliste subscribed haben, Interesse geäußert haben, dann einfach gesehn haben: [betont] ‚Oh Mann, das ist ja voll im Fluge. […] Da passiert ja so viel, da können wir nicht dran mitarbeiten'.“ (D 3) Die Integration der Entwickler gelingt trotz deren Interesse nicht. Da freie

6 Gemeint ist die Zusammenführung von parallel entwickelten Beiträgen.
7 Gemeint ist hier das Ausbleiben einer Einigung auf eine Vorgehensweise.

Softwareprojekte weder über Ressourcen verfügen, die zur Mobilisierung von neuen Mitgliedern eingesetzt werden können, noch in der Lage sind zu bestimmen, zu welchem Zeitpunkt weitere Entwickler zum Projekt hinzutreten, sind sie auf die Selbstselektion interessierter Entwickler angewiesen, deren Auftauchen sich nun aber nicht nach dem personellen Bedarf des Projekts richtet. Aus diesen Rahmenbedingungen entstehen daher besondere Anforderungen an die Integration von Entwicklern. Erstens müssen interessierten Entwicklern Beteiligungschancen angeboten werden; das Projekt muß also Möglichkeiten zur Mitarbeit und zur Übernahme von Aufgaben offerieren. Wie unser Beispiel zeigt, ergeben sich diese erst, wenn das Entwicklungsziel und das Design des Programms feststehen. Zweitens muß aber gleichzeitig damit gerechnet werden, daß sich das Interesse von neu hinzukommenden Entwicklern auf bestimmte Entwicklungsprobleme oder -ziele bezieht, die von den Gründern so nicht vorgesehen wurden. Damit also die Integration gelingt und es zu einer Mitarbeit kommt, ist eine Offenheit des Programmentwurfs gegenüber weiteren Entwicklungszielen notwendig, eine konzeptionelle Offenheit, die es den intrinsisch interessierten Entwicklern ermöglicht, zumindest in einem gewissen Rahmen eigene Ideen und Ziele zu verfolgen.

An dieser Stelle wollen wir die bisherigen Überlegungen bezüglich der sich beim Projektbeginn stellenden Anforderungen zusammenfassend zuspitzen: Zum einen stellt sich zu Beginn des Projekts die Aufgabe, ein Entwicklungsziel zu bestimmen, wobei diese Tätigkeit einen ersten Entwurf des Programms und die Vorgabe des weiteren Vorgehens einschließt. Der so entstehende Programmentwurf muß dabei kohärent sein, also ein schlüssiges Gesamtkonzept bilden. Ansonsten besteht erstens die Gefahr, daß die Entwickler ‚gegeneinander' arbeiten, also miteinander inkompatible oder unverträgliche Beiträge leisten – und zweitens, daß es an Beteiligungsmöglichkeiten für weitere Entwickler fehlt. Zum anderen setzt die Integration intrinsisch interessierter Entwickler eine Offenheit des Entwurfs gegenüber kontingenten Zielsetzungen voraus, die es ihnen ermöglicht, auf die im Projekt verfolgten Zielsetzungen Einfluß zu nehmen und ‚eigensinnige' Entwicklungsziele zu verfolgen. Es ist offensichtlich, daß diese Anforderungen in einem Spannungsverhältnis stehen und widersprüchlich sind. Stärkere Offenheit scheint nur zum Preis der Einbußen an Kohärenz zu haben zu sein. Entscheidend ist dabei für uns, daß die widersprüchlichen Anforderungen in der Sachdimension (also dem Entwurf des Programms) ihren Ursprung nicht ausschließlich im zu lösenden Entwicklungsproblem haben, sondern zu einem erheblichen Teil aus den sozialen Rahmenbedingungen resultieren, unter denen die Aufnahme der Entwicklung versucht wird.

Modularisierung

Wie werden nun diese widersprüchlichen Anforderungen bewältigt? Die Vielzahl der Ein-Personen-Projekte legt nahe, daß zumindest ein Teil nur die Hälfte der Anforderungen – die nach konzeptioneller Geschlossenheit – realisiert; die Beispiele gescheiterter Projekte – wie z.B. unser oben analysierter Fall – zeigen, daß die Hürde der Bewältigung dieser Anforderungen nicht von sämtlichen Projekten genommen wird. Im folgenden wollen wir die Strukturierungstechnik ‚Modularisierung' vorstellen, die unter erfolgreichen freien Softwareentwicklungsprojekten mit einer vergleichsweise großen Anzahl von Entwicklern recht verbreitet ist.[8] Dabei werden wir zeigen, daß Modularisierung wenn auch nicht zu einer Lösung, so doch zumindest zu einer Entschärfung der Widersprüchlichkeit führt.

Die Bedeutung der Strukturierungstechnik ‚Modularisierung' zeigt sich in einer Passage aus einem Aufsatz von Linus Torvalds, in dem er den Zusammenhang zwischen den Anforderungen der Projektstruktur und der Struktur des Quellcodes für das Linux-Projekt[9] beschreibt:

„With the Linux kernel it became clear very quickly that we want to have a system which is as modular as possible. The open-source development model really requires this, because otherwise you can't easily have people working in parallel. It's too painful when you have people working on the same part of the kernel and they clash. Without modularity I would have to check every file that changed, which would be a lot to make sure nothing was changed that would effect anything else. With modularity, when someone sends me patches to do a new filesystem and I don't necessarily trust the patches per se, I can still trust the fact that if nobody's using this filesystem, it``s not going to impact anything else." (Torvalds 1999: 108)

Deutlich kommt hier zum Ausdruck, daß die Notwendigkeit einer sachlichen Strukturierung des Quellcodes aus den sozialen Anforderungen

8 Moderne, objektorientierte Programmiersprachen sorgen per se für eine Strukturierung des Programms in unterschiedliche Module (Garzarelli 2002: 5; Bonaccorsi/Rossi 2003: 1247). Bekannte Beispiele aus dem Bereich der freien Software bilden neben dem Linux-Betriebssystem (vgl. http://www.oreilly.de/german/freebooks/linuxdrive2ger/anintro.html), der Apache-Webserver (vgl. hierzu Apache: The Definite Guide, http://www.hk8.org/old_web/linux/apache/ch15_03.htm und Franke/v. Hippel 2003: 1204) und das Grafikprogramm The Gimp (GNU Image Manipulator, vgl. http://www.faqs.org/docs/artu/ch04s04.html).

9 Linux stellt für uns hier den paradigmatischen Fall dar, da es für seine modulare Struktur bekannt ist. Sie ist ein Erbe des Betriebssystems Unix (Narduzzo/Rossi 2003) bzw. Teil der Unix-Philosophie (Raymond 2003).

des Entwicklungsmodells resultiert. Modularisierung kann unterschiedliche Grade annehmen, wobei die Entwicklung von Software in Form von offenen Programmierprojekten ein besonders hohes Maß an Zerlegung des Quellcodes erfordert. Ein hoher Zerlegungsgrad führt zur Möglichkeit, einen hohen Grad an Arbeitsteiligkeit zu verwirklichen, ein geringer Grad der Zerlegung birgt demgegenüber die Gefahr eines ‚schmerzhaften' Kollidierens von Entwicklungsbeiträgen unterschiedlicher Beteiligter. Den Anforderungen nach konzeptioneller Kohärenz bei gleichzeitiger Möglichkeit einer parallelen Entwicklungstätigkeit wird – in diesem Zitat sehr offensichtlich – durch die Einteilung des Systems in unterschiedliche Subsysteme nachgekommen.

Etwas weniger offensichtlich ist in der Passage, inwieweit sich Modularisierung auch auf die zweite Anforderung, die Offenheit gegenüber kontingenten Zielsetzungen, bezieht. Anhaltspunkte für die These, daß Modularität zu einer Offenheit gegenüber den Zielen anderer Entwickler führt, finden sich allerdings in den beiden letzten Sätzen, in denen die Übergabe von Entwicklungsbeiträgen erwähnt wird. Durch Modularisierung bleiben Veränderungen am Quellcode und ihre Folgen auf einen klar lokalisierbaren Bereich beschränkt, so daß die Unterteilung des Programms als Sicherheitstechnik wirkt. Sie ermöglicht eine einfache und schnelle Revision von unerwünschten Modifikationen des Programms und ermöglicht es damit, Freizügigkeit gegenüber den Entwicklungszielen und -beiträgen anderer Beteiligter walten zu lassen.

Um den Zusammenhang zwischen der Modularisierung und der Freizügigkeit gegenüber ‚eigensinnigen' Entwicklungszielen zu verstehen, lohnt es sich an dieser Stelle, nach den Ursprüngen dieser Technik zu fragen, die nicht im Bereich der freien Softwareentwicklung, sondern im Software Engineering liegen. Wichtige Impulse für die Idee der Zerlegung von komplexen Aufgaben in kleinere Teilaufgaben setzte David L. Parnas mit seinem klassischen Artikel „On the Criteria To Be Used in Decomposion Systems into Modules" (1972), in dem er das Prinzip des ‚information hiding' formulierte. Modularisierung – bereits im heute noch gängigen Sinne verstanden als die Zerlegung eines Systems bei eindeutiger Definition der Schnittstellen – sollte seiner Ansicht nach stattfinden, bevor die Arbeit an der Implementation der Module aufgenommen wird (ebd. 1054). Er faßt den Grad der Teilbarkeit einer Aufgabe zumindest in einem gewissen Rahmen als variabel auf[10] und geht davon aus, daß jeweils unterschiedliche Möglichkeiten existieren, ein

10 Die Gegenposition wurde in der Informatik von Frederic P. Brooks vertreten, der die Teilbarkeit der Aufgabe als gegeben und nicht von der Strukturierung der Aufgabe abhängig ansah (vgl. Brooks 1995: 18 ff.). Siehe zur Rekonstruktion der Diskussion auch Narduzzo/Rossi 2003: 8.

System in Bestandteile zu untergliedern. Hieran schließt er Überlegungen an, welches die günstigste Möglichkeit einer Zerlegung sei und entwickelt das Kriterium ‚Information hiding'. Eine Zerlegung ist demnach dann optimal, wenn die zerlegten Komponenten so wenig wie möglich über ihre innere Funktionsweise, jedoch so viel wie zu ihrer Benutzung nötig an Informationen preisgebe (ebd. 1056). Das Kriterium einer gelungenen Modularisierung ist dann erfüllt, wenn die Frage der Funktionsweise des Moduls möglichst stark von der Frage der Integration des Moduls in das Gesamtsystem getrennt wird.

Diese Konzeption von Modularisierung ist mittlerweile zum Standardrepertoire des Software Engineering geworden und findet sich in nahezu sämtlichen Lehr- und Handbüchern wieder.[11] Beispielhaft hierfür seien Pagel und Six (1994: 157) zitiert, die das Prinzip ‚Information hiding' wie folgt paraphrasieren: „Wichtig für die Benutzung einer Komponente ist, was diese im System leistet und auf welche Weise die Komponente zur Unterstützung der eigenen Aufgabe genutzt werden kann. Unerheblich ist, wie eine Komponente seine Aufgabe erfüllt." Die Überlegungen von Parnas zur Zerlegung von Systemen wurden innerhalb des Software Engineering weitergetrieben, wobei hier vor allen Dingen die Erarbeitung weiterer Qualitätskriterien für eine gelungene Modularisierung hervorzuheben ist. Zu nennen sind hier das Kriterium der Kohäsion von Modulen und ein möglichst geringes Maß an Kopplung zwischen den Modulen (Pagel/Six 175; Denert 1992: 213) im Sinne einer Minimalität von Schnittstellen (Suhr/Suhr 1993: 228; Nagl 1990: 78/79). Schnittstellen sollten dabei wohldefiniert sein, wobei damit die schriftliche Festlegung eindeutiger Spezifikationen gemeint ist (Pomberger 1993: 53; Denert 1992: 213). Zusammenfassend stellt die Modularisierung nicht einfach eine Zerlegung des Systems in kleinere Einheiten dar; ein ebenso entscheidender Bestandteil der Technik bildet die klare Schnittstellendefinition, bei kleiner Anzahl und leichter Überschaubarkeit der Modulgröße.[12] Was sind nun die Folgen einer solchen Strukturierung des Programms im Zuge des Entwurfs gemäß den Kriterien guter Modularisierung? Innerhalb des Software Engineering werden hier insbesondere drei Eigenschaften genannt, die mit der Modularisierung angestrebt werden. Dies sind erstens die Verständlichkeit des Systems, zweitens die Wiederverwendbarkeit und drittens die einfache Änderbarkeit des Systems (Suhr/Suhr 1993: 228).

Nach dieser Präzisierung können wir nun die Frage beantworten, inwieweit diese Technik die Offenheit eines Programmentwurfs gegen-

---

11 Siehe z.B. Balzert 1982: 215; Denert 1992: 214; Suhr/Suhr 1993: 235.

12 Vgl. auch die Zusammenfassung der Kriterien guter Modularisierung bei Dumke 1993: 21 und Pomberger 1993: 54.

über später hinzukommenden Entwicklungszielen beeinflußt. Durch Modularisierung steigt der Grad der Flexibilität des Programmentwurfs an. Die formale Definition von Schnittstellen erlaubt es, Programmeinheiten auszuwechseln, wobei die Funktionalität der neuen Module – zumindest in einem gewissen Rahmen – nicht der der ursprünglichen entsprechen muß. Daneben ermöglicht die eindeutige Definition von Schnittstellen eine Erweiterung des ursprünglichen Designs um zum Zeitpunkt des Entwurfs noch nicht vorhersehbare Module. Im Kontext der freien Softwareentwicklung stellt die Modularisierung nicht nur eine Antwort auf die Anforderungen nach konzeptioneller Kohärenz dar, sondern sorgt auch für die Offenheit gegenüber kontingenten Zielsetzungen – eine Voraussetzung, die wesentlich ist für eine Integration intrinsisch interessierter Entwickler. Modularisierung stellt aus diesem Blickwinkel eine Flexibilisierungstechnik von Entscheidungen in der Zeitdimension dar: Vorangegangene Entscheidungen über Ziele können schneller und einfacher durch Modularisierung revidiert werden, die Reversibilität des Programms erhöht sich und ist mit geringeren Folgekosten verbunden.

Erträge für die Technikgeneseforschung

Bis hierher haben wir gezeigt, daß die Modularität des Quellcodes eine Eigenschaft darstellt, die aus den praktischen Anforderungen des Entwicklungsprozesses resultiert. Die Anforderungen sorgen also dafür, daß die Software eine bestimmte Gestalt – nämlich eine Untergliederung des Programms in Module bei eindeutiger Definition der Schnittstellen – annimmt. Abschließend wollen wir der Frage nachgehen, wie sich dieser Befund mit den Erkenntnissen der Technikgeneseforschung verträgt.

Wie bereits weiter oben gezeigt, entspringt die Technikgeneseforschung der Auseinandersetzung mit älteren deterministischen Theorien der Technikentwicklung und stellt der Vorstellung eines technisch determinierten Entwicklungspfades die These entgegen, Technik gewinne ihre Gestalt innerhalb eines sozialen Prozesses – eine Annahme, die von sämtlichen neueren Ansätzen der Technikgeneseforschung geteilt wird. Hinsichtlich der sozialen Faktoren, mit denen die Gestalt von Technik erklärt werden soll, weisen die Ansätze jedoch große Unterschiede auf. Das Scot-Konzept[13] von Bijker/Pinch erklärt den sozialen Charakter durch Akteursgruppen, ihren Problemdefinitionen sowie den Interpretationen, wann eine Technik ein Problem löst (Pinch/Bijker 1984; Bijker

13 Social Construction of Technology (Bijker/Pinch 1987) .

1987; Bijker/Pinch 1987; zusammenfassend Hack 1995). Sozial geprägt wird Technik hiernach durch das Zusammenspiel der Interpretationen unterschiedlicher, an der Technikentwicklung beteiligter Akteure. Während dieses Konzept einer interaktionistischen Tradition zuzurechnen ist, finden sich daneben Ansätze, die sich eher makrosoziologischer Kategorien bedienen. Hierzu zählen Untersuchungen, die im Rahmen des Programms „Social Shaping of Technology" entstanden sind und die nach der Wirkung übergeordneter sozialer Strukturen wie Staat oder Ökonomie,[14] aber auch Geschlecht und Ethnizität (Dyers 1999) auf die Entwicklung von Technik fragen.[15] In diesem Punkt ähnlich, weil auf Ebene der Gesellschaftsstruktur ansetzend, ist Vorschlag von Rammert (1993), der davon ausgeht, daß Technik im Zuge der Genese unterschiedliche Orientierungskomplexe durchläuft, die konzeptionell deutliche an die Funktionssysteme der Theorien funktionaler Differenzierung angelehnt sind und die sich im zeitlichen Nacheinander gestaltprägend auf Technik auswirken. Hiervon zu unterscheiden sind wiederum Ansätze, die die Kategorie ‚Interesse' in den Mittelpunkt rücken und Technikentwicklung durch die Strategiespiele der Akteure und deren Fähigkeiten, ein das Technisierungsprojekt tragendes Netzwerk aufzubauen, erklären (Knie 1994; Hughes 1991). Der Ansatz der Arbeitsgruppe um Dierkes/Hoffmann/Marz (1992; 1998) betont hingegen den Stellenwert von Leitbildern, also Visionen oder Ideen, von denen sie annehmen, sie tragen zu einer Koordination des Handelns von Akteuren auf ein Ziel hin bei, das sich während dessen Verfolgung zunehmend konkretisiert. Daneben finden sich – quasi in einer intermediären Position zwischen Faktoren auf der Mikro- und Makroebene – Konzepte, die nach Konstruktionstraditionen fragen (z.B. Knie 1989) und deren Ursprünge in den Professionen verorten.[16]

Trotz der Buntscheckigkeit der ins Spiel gebrachten Erklärungsfaktoren ist es überraschend, daß sämtliche Konzepte ausnahmslos vom konkreten Akteurshandeln abstrahieren und die gestaltenden Kräfte nahezu überall, nur nicht im praktischen Konstruktionsprozeß selbst suchen. Zugespitzt: Die Praxis der Technikentwicklung spielt in der Technikgeneseforschung als gestalterklärender Faktor keine Rolle. Vor diesem Hintergrund tritt der innovative Aspekt unseres Gegenstands deutlich hervor. Diese Eigenschaft ‚Modularität des Quellcodes' bildet Re-

14 Vgl. zum Beispiel Hughes 1985: 45 ff. Dieser interpretiert die Arbeiten von Edison zum elektrischen Beleuchtungssystem vor allen Dingen durch ökonomische Kalkulationen geprägt. Wichtiger sozialer Einflußfaktor bei der Gestaltung dieser Technik bildet der ökonomische Effizienzgedanke.

15 Siehe hier auch zusammenfassend MacKenzie/Wajcman 1985.

16 Z.B. Schulz-Schaeffer 1996 am Beispiel der Informatiker.

sultat der praktischen Koordinationsanforderungen des Entwicklungsprozesses selbst. Als gestaltprägend erweist sich im Fall der freien Software nicht nur die Komplexität des Entwicklungsproblems, sondern mindestens ebensosehr die widersprüchlichen Anforderungen der Entwicklungspraxis, die ihre Ursache in der spezifischen sozialen Struktur offener Programmierprojekte hat. Für die Technikgeneseforschung erweist sich das Beispiel als instruktiv. Es eröffnet einen neuen Blickwinkel auf die Ursprünge der Gestalt von Technik, indem es zu den möglichen Faktoren, die auf die Gestaltung von Technik Einfluß nehmen, einen weiteren vorschlägt: die Praxis der Technikentwicklung. Es fordert dazu auf, diesen Faktor als gestaltprägend zu berücksichtigen, insbesondere dort, wo eine hohe sachliche Komplexität des technischen Artefakts in Kombination mit einer arbeitsteiligen oder verteilten Konstruktions- und Entwicklungstätigkeit zusammentrifft.

Entscheidung über Zielsetzungen: Argumentation, Reputation, Wünsche und Geschenke

In diesem Kapitel kommen wir erneut auf die klassische soziologische Frage nach den Mechanismen der Handlungskoordination zu sprechen, die hier allerdings nicht wie im vorangegangenen Kapitel mit Blick auf den Projektbeginn, sondern hinsichtlich der Definition von Zielsetzungen im Normalbetrieb freier Softwareentwicklungsprojekte gestellt wird. Diese Frage ist im Fall von offenen Programmierprojekten besonders augenfällig, da hier die Handlungskoordination unter offenbar widrigen Bedingungen stattfindet: Die Beteiligten kennen einander im Regelfall nur durch die Kommunikation auf der Mailingliste und durch ihre Arbeitsergebnisse, die Koordination der Entwicklungsbeiträge wird über die Mailingliste und nicht durch die unmittelbarere face-to-face-Kommunikation erzielt; der Kreis der Beteiligten ist offen, d.h. es können jederzeit weitere Entwickler zum Projekt hinzustoßen oder Teilnehmer ausscheiden. Daher interessieren wir uns für die Frage, wie unter diesen Bedingungen eine Entscheidung über Ziele gelingt und wie typische Muster und Lösungen dieses Koordinationsproblems aussehen.

Zu diesem Zweck werden wir die Diskussion über ein Entwicklungsziel auf der KMail-Mailingliste analysieren und in einem ersten Schritt herausarbeiten, daß Argumentation den Modus der Entscheidungsfindung bildet. Eine Einflußnahme auf Entwicklungsziele ist nur möglich, sofern der betreffende Akteur seine Position begründet. Argumentation ist allerdings ein schwacher Modus zur Herbeiführung von Entscheidungen. Dies wirft die Frage auf, wie Dissens bewältigt wird,

welche Mechanismen also im Fall des Ausbleibens von Einvernehmen greifen und eine Weiterentwicklung des Programms ermöglichen. In einem zweiten Argumentationsschritt werden wir zeigen, daß Dissens häufig zu Kompromissen führt, die hier eine typische Form annehmen: die des Einbaus von Konfigurationsoptionen.

Weiter oben haben wir herausgearbeitet, daß Beiträge zur Entwicklung von Software zu Reputation führen. Mit Blick auf das Zustandekommen von Zielen werfen die in einem Projekt existierenden Reputationsdifferenzen die Frage auf, ob und inwieweit diese den Entscheidungsprozeß beeinflussen. Bildet Reputation einen der Argumentation überlegenen Mechanismus oder stellen sie ein nur ornamentales Anhängsel ohne weiteren Einfluß auf den Entscheidungsprozeß dar? Anhand unseres Falls werden wir zeigen, daß Reputation den Koordinationsmechanismus ‚Argumentation' flankiert: Erst durch Reputation wird ein Beteiligter zu einem relevanten Anderen, mit dem Einvernehmen hergestellt werden muß, während ein solcher Zwang bei anderen, ‚reputationsarmen' Personen nicht existiert.

In einem dritten Schritt stellen wir die Frage, wie Situationen bewältigt werden, in denen Kontroversen weder durch Argumentation noch durch Kompromisse beigelegt werden können. Hier werden wir sehen, daß sich in offenen Programmierprojekten zwei Faktoren finden, die eine Stagnation der Entwicklung verhindern. Dies ist einerseits die Ermüdung der Beteiligten in Zuge anhaltender Diskussionsprozesse und andererseits die Strategie der Aufnahme der Entwicklungstätigkeit, die von den Protagonisten eines Ziels gewählt werden kann, um Blockadesituationen aufzulösen.

Abschließend werden wir die Rolle der Programmanwender bei der Definition von Zielsetzungen diskutieren. Hier zeigt sich ein uneinheitliches Bild: Die hohe Aktivität, mit der Anwender dem Projekt Entwicklungsziele vorschlagen, paart sich mit einer schwachen strategischen Position bei der Durchsetzung. Weder sind sie in der Lage, selbst die von ihnen gewünschten Ziele zu verfolgen, noch verfügen sie im Regelfall über die notwendigen technischen Kompetenzen, um in die Diskussion einzugreifen, die vorgebrachten Argumente zu beurteilen und den Entscheidungsprozeß in Richtung ihrer Wünsche zu beeinflussen. Im Projekt KMail führt das zu einem geringen Einfluß von Anwendern. Diesen Befund kontrastieren wir mit Prozessen der Zieldefinition in anderen Projekten, wie sie sich in den unserer Studie zugrunde liegenden Interviews zeigen. Hier berichten einige Interviewpartner über einen hohen Einfluß von Anwendern, den wir durch eine spezifische Form des Gabenaustauschs zwischen Entwicklern und Anwendern erklären und der entgegen einer naiven Perspektive nicht zu einer Software führt, die bes-

ser an die Bedürfnisse der Nutzer angepaßt ist, sondern die im Gegenteil z.T. dysfunktionale Nebenfolgen haben kann.

Normalmodus der Entscheidungsfindung: Argumentation

Der Fall, an dem sich die wesentlichen Aspekte der Handlungskoordination im Projekt KMail rekonstruieren lassen, betrifft Vorschläge zur Modifikation des Graphical User Interface (GUI).[1] Die Vorschläge, mit denen eine längere Diskussion über die graphische Gestaltung des Programms beginnt, stammt von einer Gruppe von Personen, die selbst nicht zu den Core-Entwicklern von KMail zählt und bislang nicht mit größeren Beiträgen zum Projekt in Erscheinung getreten ist. Dennoch kann unterstellt werden, daß die Gruppe zumindest den hoch engagierten Entwicklern von KMail vor ihrem ersten Auftritt auf der Mailingliste bekannt ist, da sie sich seit längerer Zeit im KDE-Dachprojekt engagiert hat und insbesondere der Entwickler KD 6 eine Person ist, die aufgrund ihrer Leistungen im KDE-Projekt über hohe Reputation verfügt. Die Vorschläge für die Veränderung des GUI werden in einer E-Mail von KD 6 im Namen der Entwicklergruppe an das Projekt KMail gerichtet: „Hi all, ok, I have some small but important things that we (KD 7, KD 8 and I) discussed out that we need for kmail and which I like to do and need the others approval. Sorry that KD 8 changed things this week without asking and even I didn't see what was going on." (KD 6, 2001-05-30 10:31:09) Eine Besonderheit des Vorgehens von KD 6 besteht darin, daß er sich bezüglich der geplanten Änderungen mit anderen Entwicklern abspricht. Üblicherweise wird ein Entwicklungsziel ohne weitere Vorabsprachen einfach auf der Liste des Projekts vorgeschlagen und dort diskutiert. Vermutlich möchte der Sprecher des Kollektivs

1 Hierbei handelt es sich um die graphische Benutzeroberfläche, die ein Anwender nach dem Start und während der Benutzung des Programms sieht. Es bildet also der Teil des Programms, den ein Nutzer als ‚das Programm' wahrnimmt. Wir wählen dieses Beispiel hier aus Gründen der Stringenz der Darstellung. In diesem Fall lassen sich die grundlegenden Mechanismen, die bei der Entscheidung über Zielsetzungen innerhalb des Projekts KMail eine Rolle spielen, analysieren. Allerdings sollten die Befunde nur vorsichtig auf andere Projekte generalisiert werden, da dort z.T. auch andere Formen der Handlungskoordination Anwendung finden. So wird beispielsweise dem Linux-Kernel-Projekt (vgl. die journalistisch geprägte Darstellung bei Moody 2001: 123 f.) sowie Gnome und Apache eine vergleichsweise hierarchische Form der Koordination nachgesagt, in der die Position des Projektleiters mit einem hohen Maß an Entscheidungskompetenz ausgestattet ist (Interview mit Entwickler KD 1; D 3).

hiermit die Chancen auf die Realisierung der Vorschläge erhöhen: Die Erwähnung von zwei weiteren Personen, die bei den Vorschlägen mitgewirkt haben, verleiht ihnen Gewicht und erhöht die Schwelle für eine Ablehnung oder die Formulierung von Gegenvorschlägen.

Der Umstand, daß sich KD 6 im Namen der anderen Entwickler bereit erklärt, die für diese Veränderungen zu leistende Arbeit zu übernehmen, anstatt einfach zu einer Implementation der Zielsetzung überzugehen, verweist auf eine erste Regel, an der sich die Protagonisten eines Entwicklungsziels zu orientieren haben: Sofern Entwicklungsziele eine gewisse Relevanzschwelle überschreiten, werden sie nicht einfach verfolgt, sondern im Vorfeld vor der Aufnahme der Entwicklungstätigkeit zur Diskussion gestellt, um es den anderen Beteiligten zu ermöglichen, auf die Ziele Einfluß zu nehmen. Wie gestaltet sich nun eine solche Diskussion? Ein Einblick gibt der weitere Verlauf der E-Mail, in dem die Entwicklungsziele der Mailingliste dargestellt werden:

„Now, what do we need and why do we need it ?
What we need:
the default setting should be long folder list
why: because it's the common look of mail clients and other applications having a slit view." (KD 6; 2001-05-30 10:31:09)

Der Vorschlag wird mit einer rhetorischen Frage eingeleitet, die zwei Bestandteile umfaßt: Die Frage nach dem Gegenstand des Vorschlags (‚was') und die Frage nach den Gründen hierfür (‚warum'). Bereits in dieser einleitenden Frage deutet sich an, daß im folgenden Gründe vorgebracht werden, die darauf zielen, Einvernehmlichkeit[2] in bezug auf die Zielsetzung zu erreichen. Der erste Vorschlag bezieht sich auf die grafische Erscheinung der Verzeichnisliste des E-Mail-Clients am linken Bildschirmrand,[3] die standardmäßig beim ersten Start des Programms voreingestellt ist und sich danach vom Benutzer anpassen und verändern läßt. Direkt im Anschluß an den Vorschlag wird durch ‚why' die gewünschte Veränderung der Voreinstellung begründet. Als Argument für die Veränderung bringt KD 6 vor, KMail gleiche sich so dem üblichen Erscheinungsbild anderer E-Mail-Clients an. Wir überspringen an dieser Stelle den zweiten Vorschlag aus Gründen der Stringenz und setzen unsere Analyse mit dem dritten Vorschlag des Entwicklers fort:

2 Wir führen den Begriff provisorisch ein und werden ihn unten erläutern.

3 Unter Verzeichnisse sind hier die Ablageorte von empfangener, bearbeiteter und gesendeter E-Mail gemeint. Standardmäßig legt KMail im Zuge der Installation die Verzeichnisse Inbox, Outbox, Sent-mail, Trash und ‚Drafts' an (vgl. das Benutzerhandbuch von KMail: http://docs.kde.org/en/3.3/kdepim/kmail).

„c) more columns in the folder view and the mail view for various purposes which ones: a column in the folder view for the unread mails and one for the total mails, just like knode.
why: this is pretty standard and has proved to be efficient towards the user looking at the folder view, also makes the clients look more consistent.
Another one in the listbox of the mails to sort threaded/unthreaded. I know that can be done via the menu or the configuration but even I had to look very hard for that feature to find it.“ (KD 6; 2001-05-30 10:31:09)

Auch hier folgt der Autor der Struktur der Nennung des Entwicklungsziels und einer sich daran anschließenden Begründung. Die Begründung der Einführung der vorgeschlagenen Kolumnen in der Verzeichnisliste erfolgt wiederum durch den Verweis auf ein Programm, das als Vorbild fungiert, wobei das Argument hier im Vergleich zum ersten Vorschlag weiter ausgeführt wird: Eine Orientierung am Vorbild stellt KD 6 folgend kein Selbstzweck dar, sondern soll zur Übernahme einer effizienten und bewährten Funktionalität führen. Er beendet seine Vorschläge mit einem Ausblick auf das von ihm gewünschte weitere Vorgehen:

„We would like to have these little changes done for 2.2 and would like to do them with you guys together as we think these are needed GUI improvements that would make kmail look *a lot* better and make it much easier for beginners to handle it. Please feel free to comment and blame me if something goes wrong if you're also up with these ideas.“ (KD 6; 2001-05-30 10:31:09)

In dieser abschließenden Passage nennt der Autor einen Zeitpunkt, zu dem er die Implementation der Vorschläge anstrebt. Mit dem Kürzel 2.2 ist eine Version von KDE gemeint, die zu diesem Zeitpunkt den nächsten Major-Release darstellt, der sich an einen großen Kreis von Benutzern richtet.[4] Mit der kommunizierten Bereitschaft, sich am Projekt zu beteiligen, qualifiziert sich der Beitrag nicht als Wunsch oder Feature-Request, sondern als echter Vorschlag. Interessant an dieser letzten Passage ist die an andere Beteiligte gerichtete Aufforderung, die Vorschläge zu kommentieren. Diese unterstützt unsere obige Interpretation, derzufolge Entwickler sich um die Herstellung von Einvernehmlichkeit bemühen. Zentraler Aspekt der Orientierung an Einvernehmlichkeit bildet die hier gezeigte Bereitschaft, sich mit der Meinung anderer Projektteilnehmer zu beschäftigen und mit ihnen in einen Diskussionsprozeß zu treten. Weiter bemüht sich KD 6 um die Vermeidung unnötiger Arbeit,

4 Die Veröffentlichung von KDE 2.2 war am 15. August 2001. Die Version wurde vom Projekt als eine „more polished, better integrated and more feature-rich desktop experience“ angekündigt (vgl. http://kde.org/announcements).

indem er nicht nur die anderen Entwickler von seinem Vorhaben in Kenntnis setzt, sondern sie auch dazu auffordert, sich zu Wort zu melden, sofern bereits an der Realisierung der vorgeschlagenen Funktionalität gearbeitet wird.

Die hier ausschnitthaft interpretierte E-Mail führt auf der KMail-Mailingliste zu unterschiedlichen Reaktionen und bildet den Ausgangspunkt für eine ausführliche Diskussion der Vorschläge. Deren Intensität erklärt sich nun einerseits dadurch, daß die Entwickler mit ihrer Bereitschaft zur Implementation einen Handlungsdruck erzeugt haben. Andererseits beziehen sich die Vorschläge auf einen wichtigen Bestandteil, die grafische Erscheinung des Programms, durch deren Sichtbarkeit und Präsenz davon ausgegangen werden kann, daß hier sämtliche Entwickler Präferenzen besitzen.

Während ein Teil der Reaktionen anderer Entwickler Nachfragen darstellen, die dann Erläuterungen, Explikationen und Präzisierungen der Zielsetzungen von KD 6 nach sich ziehen, führt insbesondere der Vorschlag ‚c)', die Implementation weiterer Spalten, zu einer Kontroverse. So kommentiert ein Entwickler diesen Vorschlag wie folgt:

„I think you should be able to turn that off, though. I don't think it's possible with the kmail version from kde-2.1.1 to delete columns but I think that would really be a good idea. You could then add as much columns as you want without doing something wrong. You'd have to talk about the default setting though." (KD 9; 20012-05-30 10:49:42)

Der Entwickler KD 9 greift die Überlegungen zur Veränderung des GUI auf, allerdings um nun seinerseits einen alternativen Vorschlag zu entwerfen. Er möchte Konfigurationsmöglichkeiten einrichten, so daß der Anwender selbst entscheiden kann, welche Spalten angezeigt werden. KD 9 orientiert sich damit nicht – wie KD 6 im ursprünglichen Vorschlag – am Kriterium einer einfachen Verwendbarkeit des Programms für neue Anwender und eines leichten Zugriffs auf die Funktionalität, sondern an dem der Handlungsfreiheit des Benutzers. An diese Umdefinition des Entwicklungsziels anschließend kämpft der Entwickler KD 6 um seine ursprüngliche Idee – vom Auftauchen eines konkurrierenden Entwicklungsziels sichtlich wenig begeistert führt er aus:

„Hmm… I think changing the default by itself without making that configurable does make the most sense. Please have a look at knode[5] for what I mean

5 Newsreader für die Benutzeroberfläche KDE (siehe auch die Webseite des Projekts http://knode.source forge.net/whatis.php).

(nsmail and outlook express do the same as pretty every mail client around)
[…]
What I want to have it to look like this:
column1: Foldername columne2: number of unread mails column 3: number of total mails in folder. That's the precise look :).“ (KD 6; 2001-0530 11:35:09)

In diesem Abschnitt findet sich erneut ein Verweis auf andere Programme, in denen die Spalten so implementiert sind, wie von KD 6 in der vorangegangenen E-Mail vorgeschlagen. Es wird hier sehr deutlich, daß er durch andere Programme dazu inspiriert wurde, dieses Entwicklungsziel vorzuschlagen, so daß es sich um eine imitative Innovation, also dem horizontalen Ideentransfer von einem Programm zu einem anderen handelt. Nachgeahmt werden soll dabei nicht eine bestimmte Funktionalität, sondern die Gestaltung der grafischen Benutzeroberfläche und den Zugriff auf die Funktionalität des Programms im Sinne eines Look-and-Feel. Auch hier dauert es wiederum nur wenige Minuten, bis KD 9 auf diese Replik antwortet. Der oben beschriebene Eindruck, daß die Entwickler bei der Auswahl und Beurteilung von Entwicklungszielen unterschiedliche Kriterien anwenden, verdichtet sich hier:

„I don't understand why you feel that you should take the choice of what the user wants out of his hands. That is IMO[6] pretty stupid. Sure, the default is very important as most beginners don't change it but if the user KNOWS what he wants then he should be able to do it.
> column1: Foldername columne2: number of unread mails column 3: number of total mails in folder. That's the precise look :)
Now that I understand it I think it's a good idea“ (KD 9; 2001-05-30)

KD 9 betont erneut die aus seiner Sicht hohe Relevanz des Kriteriums der Anpaßbarkeit des Programms an die Bedürfnisse der Benutzter, es liegt ein Dissens hinsichtlich der anzustrebenden Entwicklungsziele vor, der seine Ursache in grundlegenderen Bewertungskriterien – Anpaßbarkeit vs. leichte Eingewöhnung von ‚Umsteigern' – hat. Während in diesem ersten Teil der E-Mail von KD 9 der ursprüngliche Vorschlag der Veränderung des GUI von KD 6 abgelehnt wird, überrascht es, daß er gegen Ende dem Vorschlag unter der Bedingung des Einbaus einer Konfigurationsoption zustimmt.

Einverständnis zu diesem Vorschlag wird auch von einem anderen Entwickler durch eine E-Mail signalisiert, was zur Folge hat, daß sich nach diesen beiden unterstützenden Beiträgen KD 6 erneut zu Wort

6 Akronym für ‚in my Opinion'.

meldet und den bisherigen Entscheidungsprozeß der erst vier Stunden alten Diskussion wie folgt zusammenfaßt:

„Ok, that[7] was probaly too drastic. We can make it configurable with a checkbox like ‚use old KMail user interface' or something
>> column1: Foldername columne2: number of unread mails column 3: number of total
>> mails in folder. That's the precise look :)
> Now that I understand it I think it's a good idea
ok, then we have agreed on this as well" (KD 6; 2001-05-30 14:20:55)

Aufgrund des Widerspruchs sieht KD 6 eine Konfigurationsoption vor, die es ermöglicht, das zu diesem Zeitpunkt noch existierende KMail-GUI zu nutzen. Die Standardeinstellung des neuen GUI solle dann dem ursprünglichen Vorschlag von KD 6 entsprechen. Ein neues Entwicklungsziel scheint gefunden zu sein und es sieht hier so aus, als sei der Entscheidungsprozeß abgeschlossen.

Die Erträge unserer Beschäftigung mit dem Fall des Graphik User Interface läßt sich wie folgt zusammenfassen: Erstens orientieren sich die Entwickler im Fall von Modifikationen mit hohem Stellenwert an den Normen von Transparenz und Offenheit des Entscheidungsprozesses. Vorschläge werden zur Diskussion gestellt, Entscheidungssituationen als solche kenntlich gemacht, was es anderen Mitgliedern des Projekts ermöglicht, sich an der Diskussion zu beteiligen. Dies hatte sich u.a. bereits in der ersten E-Mail gezeigt, in der der Vorschlag zur Modifikation des GUI formuliert wurde, am Beispiel der Entschuldigung für Veränderungen, die vorher nicht mit den Entwicklern des Projekts abgesprochen wurden – und an der expliziten Aufforderung zu Stellungnahme zu den Vorschlägen. Zweitens haben wir gesehen, daß Entwicklungsziele nicht einfach vorgeschlagen und erläutert, sondern regelmäßig argumentativ begründet werden. Sie werden typischerweise durch grundlegendere Bewertungsnormen gerechtfertigt, wobei der Erfolg des Programms, der sich in Popularität ausdrückt, oder die Handlungsfreiheit des Benutzers hierfür Beispiele darstellen. Drittens zielt die Argumentation in diesen technischen Kontroversen nicht auf die Herstellung eines Konsensus über Zielsetzungen, Bewertungsnormen und Schlußregeln, sondern – wesentlich bescheidener – auf die Herstellung einer Situation, in der sich kein weiterer Widerstand gegen die vorgeschlagenen

7 Gemeint ist hier der ursprüngliche Vorschlag von KD 6, das GUI zu modifizieren, ohne eine Konfigurationsoption einzuführen.

Entwicklungsziele formiert.[8] Das Ziel der Diskussion bildet die Herstellung von Einvernehmlichkeit, wobei wir damit die Abwesenheit von offenem Protest in Form von ‚exit' und ‚voice'[9] meinen. Einvernehmen bildet die Voraussetzung dafür, daß eine Weiterentwicklung möglich ist und es weder zu einer Aufspaltung des Projekts noch zu miteinander unverträglichen Entwicklungstätigkeiten kommt. Unser Fallbeispiel zeigt, daß der Kompromiß, bei dem keine der Parteien ihre Maximalforderungen durchsetzen kann, das Mittel zur Herstellung von Einvernehmlichkeit darstellt. Wie dieser aussieht, ist vom Verhandlungsgeschick der Beteiligten und der Schlagkräftigkeit der vorgebrachten Argumente anhängig.

Der Vergleich mit anderen Kontroversen über Zielsetzungen im Bereich der freien Softwareentwicklung zeigt viertens, daß es bei der scheinbar ad hoc entwickelten Lösung – Einbau einer Konfigurationsoption – um einen typischen Kompromiß handelt, der seine Ursache in Rahmenbedingungen freier Softwareentwicklungsprojekte hat. Wie schon mehrfach angesprochen, zeichnen sich offene Programmierprojekte dadurch aus, daß ihre Beteiligten intrinsisch motiviert sind und sich ihr Interesse auf bestimmte Entwicklungsprobleme oder die Implementation einer bestimmten Funktionalität bezieht. Dies führt zu einem großen Bündel von im Projekt verfolgter Zielsetzungen, die nun allerdings nicht von sämtlichen Entwicklern gleichermaßen erwünscht sind. Die Implementation einer Funktion bei gleichzeitiger Konfigurierbarkeit bildet einen Kompromiß, der sowohl dem Wunsch nach Verfügbarkeit als auch nach Abwesenheit der Funktionalität Rechnung trägt: Konfigurationsoptionen ermöglichen es, daß gewünschte Funktionalitäten zur Hand sind und nicht benötigte in den Hintergrund treten – und kommen so den Wünschen sämtlicher Beteiligter entgegen. Allerdings hat die Schließung technischer Kontroversen mittels der Lösung ‚Konfigurationsoption' auch ihre Kosten: Wenn die zur Verfügung stehende Funktionalität, der Zugriff darauf, die Erscheinung des Programms sowie die Einpassung des Programms in das Gesamtsystem veränder- und anpaßbar wird, der Kompromiß also zu häufig gewählt wird, droht die Gefahr, daß die Konfiguration des Programms gerade für weniger erfahrene Anwender unübersichtlich wird. Fünftens haben wir gesehen, daß eine wichtige Quelle zur Generierung von Zielen die Orientierung an anderen Projekten darstellt. Sie werden z.T. dadurch gewonnen, daß andere Programme analysiert und Funktionalitäten dort ‚abgekupfert' werden. Eine

8 Daher stellen freie Softwareentwicklungsprojekte auch kein Beispiel für Konsenstheorien unterschiedlicher Provenienz (wie z.B. Habermas 1973: 251 ff. oder Miller 1992) dar.

9 Im Sinne von Hirschman 1970.

starke Orientierung an Vorbildern ist insbesondere dann zu erwarten, wenn ein Programm die eindeutige Marktführerschaft innehat. Wie wir anhand der Begründung von KD 6 gesehen haben, dient imitative Programmentwicklung dann dem Ziel der Popularisierung des eigenen Programms durch die Übernahme erfolgsverursachender Merkmale vom Vorbild.

Der Einfluß von Reputation

Weiter oben hatten wir festgestellt, daß unter den Entwicklern ein Reputationssystem existiert, das Gaben – also die Beiträge zur Entwicklung – mit dem Erwerb eines Rufs belohnt. An dieser Stelle wollen wir nun fragen, welche Rolle Reputation bei Entscheidungen über Entwicklungsziele spielt. Gelingt es, aufgrund von vorangegangenen Leistungen in einem höheren Maße Einfluß auf die verfolgten Zielsetzungen auszuüben; stellt Reputation einen konkurrierenden oder komplementären Mechanismus der Entscheidungsfindung dar? Aufschluß darüber gibt uns wiederum der Fall des GUI, der mit der oben zitierten E-Mail nicht an seinem Ende angekommen ist. Ein hoch engagiertes Mitglied des Projekts meldet sich einige Stunden nach dem Kompromiß zu Wort:

[…]
> yes. Agreed with Magnus as well, so I think this is the best choice to
> go and have the new behavior the default toward the user
Hi, I didn't say anything about several columns. I prefer the way it is currently. When I don't have any unread mails, then I also don`t need an empty column for their number. […] At least I like to have as few columns as possible to not waste space with unimportant things.“ (KD 10; 2001-05-30, 18:09:49)

Mit der Feststellung, er habe sich noch nicht zum vorgeschlagenen Ziel geäußert, positioniert sich KD 10 als relevanter Akteur. Gleiches gilt für die Begrüßungsformel ‚Hi‘, die hier nicht am Beginn, sondern etwa in der Mitte der E-Mail auftaucht, nachdem sich der Autor bereits zu anderen Aspekten der Diskussion geäußert hat. Er sieht sich als übergangen und den Entscheidungsprozeß keineswegs als abgeschlossen an. Durch die Positionierung (und die weiter unten akzeptierende Reaktion von KD 6 darauf) wird deutlich, daß er eine Person ist, auf dessen Einvernehmen es bei der Entscheidung über Entwicklungsziele ankommt.

Aber auch dieser Entwickler lehnt das von KD 6 vorgeschlagene Entwicklungsziel nicht Kraft seiner Position als eine zentrale Figur im

Projekt oder qua seiner hohen Reputation ab, sondern – ebenso wie sämtliche anderen Mitglieder – auf der Basis von Argumenten. Weder Reputation noch Engagement entbindet also von der Verpflichtung zur Begründung einer Position. In der nun folgenden Replik des Entwicklers KD 6 nimmt dieser Bezug auf die Sonderrolle des Maintainers.

„Yes, well, agreed you're the maintainer, that gives your personal preference a great influence in the behavoir. I agree with you that this might be true for some users, especially long-term kmail and unix users. But if you want to get windows users to use it, the default has to be different and, most important, consistent with knode which orients itself on the ‚standard' userinterface." (KD 6; 2001-05-30 18:53:26)

Der Entwickler KD 6, der sich an dieser Stelle immer noch um die Durchsetzung seiner Entwicklungsziele bemüht, räumt ein, dem Entwickler käme aufgrund seiner zentralen Position ein hoher Einfluß auf das Verhalten des Programms zu. Er nimmt also den Widerspruch zur Kenntnis und gibt – wie man vielleicht erwarten könnte – nun nicht das Entwicklungsziel auf, sondern begründet es noch einmal ausführlich.

Was läßt sich nun aus diesem Beispiel für das Verhältnis von Argumentation einerseits und Reputation andererseits bei der Herbeiführung von Entscheidungen innerhalb von freien Softwareprojekten lernen? Die Intervention erfolgt zu einem Zeitpunkt, an dem der Entwickler KD 6 versucht, den Diskussions- und Entscheidungsprozeß zu einem Ende zu bringen. Die hohe Reputation des Entwicklers führt nicht zu einer Ablehnung des Vorschlags von KD 6, sondern zu einer erneuten Aufnahme des Austauschs von Argumenten. Die ‚Begründungspflicht' als Kennzeichen der Argumentation gilt dabei für sämtliche Beteiligte – einschließlich Personen, die über hohe Reputation verfügen.

Der Umstand, daß der Maintainer in der Lage ist, den Entscheidungsprozeß zu einem weit fortgeschrittenen Zeitpunkt der Diskussion zu öffnen und den Protagonisten dazu zu bringen, seine Position erneut zu begründen, macht nun deutlich, welche Rolle Reputation im Entscheidungsprozeß spielt: Von der Höhe der Reputation ist es abhängig, wie wichtig das Einvernehmen mit dem betreffenden Akteur ist. So führt hohe Reputation zu einer hohen Relevanz von Einvernehmen mit der betreffenden Person während geringe Reputation – wie im Fall eines sporadischen Entwicklers oder eines Anwenders, der sich an der Diskussion auf der Mailingliste beteiligt – zu einer niedrigen Relevanz des Einvernehmens führt. Über Dissens bezüglich von Entwicklungszielen mit Personen, die über geringe Reputation verfügen, kann, sofern Personen mit hoher Reputation Einvernehmen signalisieren, hinweggegangen werden,

während dies im Fall von Personen von hoher Reputation nicht denkbar ist. Kurz: Reputation führt dazu, im Zuge von Entscheidungsprozessen über Ziele zu einem relevanten Anderen zu werden.

Entscheidungsschwäche von freien Softwareentwicklungsprojekten?

Im Vergleich zur Hierarchie, bei der Entscheidungskompetenzen klar verteilt sind und das Entscheiden ranghöherer Personen qua ihrer Position für rangniedrigere Personen bindenden Charakter hat, verfügen offene Programmierprojekte über einen schwachen Modus der Entscheidungsfindung: Es ist denkbar, daß Kontroversen über Zielsetzungen nicht vermittels des Austauschs von Argumenten zu lösen sind und Einvernehmlichkeit ausbleibt. Naheliegend wäre es demnach, von einer Entscheidungsschwäche freier Softwareentwicklungsprojekte auszugehen, die offensichtlich wird, sobald Argumentation scheitert, der Kompromiß ‚Konfigurationsoption' abgelehnt wird und eine Blockade der weiteren Entwicklung eintritt. Solche Situationen sind nun allerdings recht selten,[10] da sich in offenen Programmierprojekten zwei Faktoren finden, die das Verharren in einer stagnativen Situation verhindern und Blockadesituationen aufsprengen. Bei beiden handelt es sich nicht um argumentative Formen der Bewältigung von offenem Dissens, die auf der Projekt-Mailingliste zu beobachten wäre, sondern um solche, die sich eher lautlos im Hintergrund vollziehen. Aus diesem Grund verlassen wir an dieser Stelle unser Beispiel der Überarbeitung des GUI und analysieren im folgenden eine Passage aus einem Interview mit einem KMail-Entwickler, in der dieser die Frage, ob Kontroversen zur Blockade der Entwicklung führen, wie folgt beantwortet:

„Nee, langfristig nicht. Also, es gibt dann einen Thread, der 50 oder so Messages umfassen kann. […] Das geht dann im Ernstfall anderthalb Wochen oder so, geht das hin und her. Und irgendwann sind dann die Leute angenervt genug, daß sie sich dann zusammenraufen und dann irgendwas zusammenbauen. Es kann sein, daß der eine oder andere dann im Hintergrund ein bißchen unzufrieden darüber ist, aber man kann's nicht jedem Recht machen." (KD 1)

Längere Diskussionen führen nicht zwangsläufig dazu, daß eine Partei die andere überzeugt, sondern es kommt irgendwann zu einer Diskussi-

10 Weder konnten Stagnationen der Entwicklung im Untersuchungszeitraum auf der Mailingliste des Projekts beobachtet werden, noch berichteten die Entwickler in den Interviews von derartigen Situationen.

onsmüdigkeit. Die Beteiligten sind intrinsisch an der Entwicklungstätigkeit interessiert, nicht aber an andauernden Diskussionen und erst recht nicht an solchen, die zu keinem Ergebnis führen. Die Chancen für das Entstehen von Einvernehmlichkeit steigen also mit der Dauer der Diskussion, da ein Beharren auf der Position weder den Entscheidungsprozeß noch die Entwicklung des Programms weiterbringt. Entscheidungsbefördernd ist also nicht nur die Überzeugungskraft der vorgebrachten Argumente, sondern auch der Umstand, daß es sich dabei um eine Tätigkeit handelt, die zur Ermüdung der Beteiligten und der unbeteiligten Leserschaft auf der Mailingliste führt. Die Aufgabe einer Position, ohne von den Argumenten des Kontrahenten überzeugt zu sein, führt zwar zu Unzufriedenheit, ermöglicht aber gleichzeitig ein Weiterkommen im Projekt.

Neben der Ermüdung der Diskussionsteilnehmer gibt es allerdings noch einen weiteren Faktor, der Blockadesituationen aufbricht. Exemplarisch sei hier aus dem Interview mit dem Entwickler KD 2 zitiert, der die Lösung von lang anhaltenden Kontroversen über Entwicklungsziele wie folgt beschreibt:

„Ja und vor allem ist keiner da, der dann wirklich sagt, so wird's gemacht, ihr könnt euch jetzt nicht entscheiden. Im Extremfall ist es der, der dann an den Editor geht und einen Patch schreibt. Der sich dann wirklich die Arbeit macht und nicht nur rumlabert." (KD 2)

Das Fehlen einer Entscheidungsinstanz, die Kontroversen über Zielsetzungen beendet, wird dadurch kompensiert, daß vom Diskussionsprozeß zur Entwicklungstätigkeit übergegangen werden kann, ohne Einvernehmlichkeit über die verfolgten Ziele erreicht zu haben. Aufschlußreich am Zitat ist dabei, daß die Entwicklungstätigkeit hier sehr praktisch-anschaulich durch die Formulierungen ‚sich Arbeit machen' und ‚an den Editor gehen' beschrieben, während die Beteiligung am Diskussionsprozeß hier abwertend als ‚Rumlabern' bezeichnet wird. In dieser Kontrastierung der beiden Tätigkeiten kommt eine höhere Wertschätzung der praktischen Entwicklungstätigkeit – also der Arbeit an der Verwirklichung von Zielsetzungen – gegenüber Beiträgen zur Diskussion zum Ausdruck. Entscheidend ist, daß vor der Wahl dieser Option allerdings die Diskussion über das Ziel gesucht und damit der Versuch unternommen wird, auf dem Weg der Argumentation Einvernehmlichkeit über Zielsetzungen herzustellen. Das Mittel ist also erst dann legitim, wenn die argumentativen Möglichkeiten einer Herbeiführung von Entscheidungen ausgeschöpft sind. Evidenz für unsere These, daß die Vorrangigkeit des ‚Primats der Implementation' vor dem ‚Primat der Argumen-

tation' ausschließlich in Konfliktfällen gilt, findet sich auch in unserem Fallbeispiel der Modifikation des Graphic User Interface: Hier hatte ein Entwickler mit der Implementation begonnen, ohne vorher mit den Beteiligten des Projekts in einen Diskussionsprozeß getreten zu sein – ein Verhalten, das einer Entschuldigung bedurfte.

Benutzergetriebene, partizipative Innovation?

Nach der Analyse der grundlegenden Mechanismen wollen wir unser Verständnis zur Entscheidungslogik von offenen Programmierprojekten durch die Beantwortung der Frage vertiefen, welche Rolle Anwender im KMail-Projekt bei der Definition von Zielen spielen und wie deren Beitrag zu beurteilen und einzuschätzen ist. Im ursprünglichen Konzept der Innovationsnetzwerke spielen die Anwender an dieser Stelle eine außerordentlich große Rolle: Ebenso wie die Hersteller von Technik beteiligen sie sich an Aushandlungsprozessen, in denen in rückgekoppelten Lernprozessen Ziele generiert werden, und leisten so einen wichtigen Beitrag zu Innovationen.[11] Finden sich nun ähnliche Prozesse im Bereich der freien Softwareentwicklung? Wird die Transparenz des Entwicklungsprozesses und der Erreichbarkeit der Programmierer von Anwendern dazu genutzt, Zielsetzungen maßgeblich zu beeinflussen, so daß die Programme den Bedürfnissen der Benutzer besser entsprechen? Weiter gefragt: Lösen offene Programmierprojekte durch die Möglichkeiten einer Beteiligung die in der Literatur zur partizipativen Technikentwicklung und partizipativen Technikfolgenabschätzung geäußerten Hoffnungen nach einer egalitaristischen (Vatter 1998), besser legitimierten, besser akzeptierten (Bechmann 1997), sozial und ökologisch verträglicheren (Wienhöfer 1996) sowie rationaleren (Köberle/Gloede/Hennen 1997) Technik ein? In der Literatur zur freien Softwareentwicklung wird die Beteiligung von Anwendern an der Entwicklung von freier Software durchaus bemerkt, wobei Beteiligung hier in ähnlicher Weise stark positiv attribuiert wird[12] wie in der sozialwissenschaftlichen Literatur. Im folgenden werden wir eine weniger optimistische Perspektive auf die Anwenderpartizipation in freien Softwareentwicklungsprojekten entfalten. Die Beobachtung des geringen Einflusses erklärt sich vorrangig durch die ungünstige strategische Position, in der sich Anwender befinden. Wir werden zeigen, daß einzig ein Gabenaustausch zwischen Entwicklern und Anwendern zu einem gewissen Einfluß Letzterer auf Ziel-

11 Kowol/Krohn 1997: 53; Asdonk/Bredeweg/Kowol 1991: 292 f., 1994: 77.
12 Vgl. z.B. Healy/Schussman 2003: 5.

setzungen führt, der aber nicht immer vorzufinden ist und dysfunktionale Effekte haben kann.

Empirische Unterstützung für die These einer besseren, da in einem partizipativen Prozeß entwickelten freien Software findet sich bei Franke/v.Hippel (2003). Diese kommen bei einer Untersuchung unter Anwendern zur Zufriedenheit mit dem Apache-Webserver zu dem Schluß, daß User, die sich an der Entwicklung des Programms beteiligt und im Rahmen des Projekts selbst Ziele verfolgt haben, mit dem Programm in einem höheren Maße zufrieden sind als solche, die keinen Beitrag zur Entwicklung geleistet haben. Die Anpaßbarkeit des Programms durch eigene Entwicklungsbeiträge sei den Autoren folgend ein wichtiger Faktor, der es erlaube, den heterogenen Bedürfnissen Rechnung zu tragen. Allerdings handelt es sich im Fall des Webservers nun um ein Programm, daß sich an einen sehr speziellen Anwenderkreis richtet. Er wird üblicherweise von Organisationen genutzt, die das System für ihren Internet-Auftritt einsetzen. In dem von Franke/v.Hippel befragten Sample sind so auch 76 Prozent der Anwender For-Profit-Organisationen, und ein großer Teil der Befragten gibt an, über die zur Modifikation des Programms notwendigen Kompetenzen zu verfügen (ebd. 1209), oder äußert die Bereitschaft, für eine Anpassung des Programms zu bezahlen.[13] Die Befunde lassen sich daher nicht umstandslos auf andere Projekte übertragen, bei denen die Anwender nicht in ähnlicher Weise über die Möglichkeit einer Anpassung des Programms verfügen. Da die Aneignung von Programmierkompetenzen nicht kurzfristig mit niedrigem Aufwand zu realisieren ist, ergibt sich für einen großen Teil der Nutzer eine völlig anders gelagerte Situation: Sie können nicht – wie die Nutzer in der oben zitierten Untersuchung – die Grenze zwischen Anwendern und Entwicklern queren, um zu (sporadischen) Entwicklern zu werden oder Entwickler zur Anpassung des Programms bezahlen. Um diese Unterschiede konzeptionell einzufangen, bietet es sich an, zwischen verschiedenen Formen der Beteiligung zu unterscheiden und anhand der Differenzen die sich für die Akteure ergebenden strategischen Positionen zur Beeinflussung von Zielsetzungen zu bestimmen.

Instruktiv ist hier, ausgehend vom Begriff der Öffentlichkeit[14] in Anschluß an Bora zwischen zwei Ausprägungen zu unterscheiden: Publizität und Partizipation. Publizität meint, daß Ereignisse und Informationen transparent und zugänglich sind, Partizipation die Möglichkeit ei-

13 Die Zahlungsbereitschaft der Untersuchung fällt mit durchschnittlich 5232 US-Dollar erstaunlich hoch aus. Dieser Befund läßt sich nur durch die hohe Verbreitung des Programms im professionellen Bereich erklären.

14 Wie wir weiter oben gesehen haben, stellt Öffentlichkeit ein wesentliches Merkmal von offenen Programmierprojekten dar.

ner Teilnahme, also die Chance der Einnahme von Leistungsrollen (1999: 73). Das Prinzip der Publizität hält die Rolle des Zuschauers bereit, die durch das Fehlen von ‚voice'- und ‚exit'-Optionen gekennzeichnet ist. Von Partizipation ist hingegen dann zu sprechen, wenn weitergehende Formen einer Beteiligung existieren, also Rollen zu besetzen sind, die ‚exit'- und/oder ‚voice'-Optionen beinhalten. Mit Blick auf offene Programmierprojekte können wir diesen Ausgangspunkt in Richtung einer abgestuften Typologie der Inklusion ausbauen.

Eine erste Einsicht gewinnen wir, indem wir uns den im vorangegangenen Abschnitt herausgearbeiteten Modus der Entscheidungsfindung vergegenwärtigen. Obwohl sich prinzipiell jedermann auf der Mailingliste zu Wort melden kann, sieht der Entscheidungsprozeß nur für Personen, die über hohe technische Kompetenzen verfügen, Leistungsrollen vor: Positionen müssen argumentativ belegt, für Kritik Gründe angeführt werden. Die Beteiligung an der Diskussion setzt damit ein erhebliches Kompetenzniveau voraus, das Anwender meist nicht erreichen. Die ‚voice'-Option ist nur Akteuren mit hoher technischer Kompetenz verfügbar, also den Core-Entwicklern, den sporadischen Entwicklern und Anwendern, die ohne größeren (Lern-)Aufwand zu sporadischen Entwicklern werden könnten.[15] Von der ‚voice'-Option ausgeschlossen bleiben hingegen Anwender mit geringer technischer Kompetenz. Dies führt zu einer sozialen Abschließung von Entscheidungsprozessen. Exklusiver als ‚voice' ist die ‚exit'-Option: Zwar verfügen sowohl die Entwickler als auch die Anwender aufgrund der niedrigen Ausstiegsbarrieren[16] jederzeit über die Möglichkeit, das Projekt zu verlassen, als Mittel des Protests gegen Entscheidungen kann ‚exit' jedoch nur von Personen angewandt werden, deren Beträge herausragend sind und deren Ausscheiden für das Projekt einen Verlust bedeuten würde. Dies ist nun allerdings weder bei Anwendern – deren Abwanderung unter die Aufmerksamkeitsschwelle des Projekts fällt – noch bei sporadischen Entwicklern, unter denen eine hohe Fluktuation anzutreffen ist, der Fall. Da das Gros der Arbeit in offenen Programmierprojekten von einem kleinen Kreis von Core-Entwicklern[17] geleistet wird, ist diese Form des Widerstands gegen Entscheidungen dieser Gruppe vorbehalten.

---

15 Vgl. hierzu auch Raymond 1999: 36. Dieser sieht als Vorteil einer großen Nutzerbasis an, daß ein Teil der Anwender zu Co-Entwicklern werden kann. Der Übertritt von der Gruppe der Anwender zu der der Entwickler ist ein weitverbreitetes Phänomen, das sich auch in den dieser Arbeit zugrunde liegenden Interviews immer wieder zeigt.

16 Hierzu ist lediglich eine schlichte E-Mail mit dem Betreff ‚unsubscribe me' an die Mailingliste des Projekts notwendig.

17 Siehe hierzu Ghosh/Robles/Glott 2002: 20.

Es existieren also innerhalb von offenen Programmierprojekten verschiedene Formen der Beteiligung, wobei der Zugang zu Leistungsrollen durch Kompetenz reguliert wird. (1) Am Pol Publizität finden sich Anwender, die nicht über Kompetenzen verfügen, um an der Diskussion von Zielsetzungen teilzunehmen, und deren Abwanderung – gerade in großen Projekten – kaum zur Kenntnis genommen wird. (2) In einem stärkeren Maße beteiligt sind Anwender oder sporadische Entwickler, die ein gewisses Kompetenzniveau besitzen und die damit über ‚voice'-, nicht aber über ‚exit'-Optionen verfügen, um zu protestieren. (3) Einzig die Core-Entwickler sind Akteure, die in für das Projekt konsequenzreicher Weise abwandern können und für die faktisch auch eine zweite Form der Artikulation von Protest besteht. Nach diesen Vorüberlegungen zur strategischen Position der unterschiedlichen Akteure können wir auf die Frage zurückkommen, ob Anwender, die selbst nicht mit vertretbarem Aufwand zu Entwicklern werden können, Einfluß auf die Definition von Entwicklungszielen ausüben und ob dieser tatsächlich so gering ist, wie diese Vorüberlegungen nahelegen.

Eine auffällige Form der Einflußnahme auf die Zielsetzungen im Projekt KMail bilden *Wünsche (wishes)*, die hauptsächlich von Anwendern vorgebracht werden. Ein Wunsch stellt eine Handlungsaufforderung dar, mit der Ego Alter dazu motivieren möchte, eine Leistung zu erbringen, wobei Ego keine direkte Gegenleistung offeriert. Damit der Wunsch also erfüllt und – in unserem Fall – das Entwicklungsziel verwirklicht wird, ist der Anwender darauf angewiesen, daß der Handlungsaufforderung freiwillig nachgekommen wird. Wie bereits einleitend beschrieben, verfügt das Projekt KMail über ein Bug-Track-System, vermittels dessen Wünsche in einer separaten Liste gesammelt und verwaltet werden. Das Projekt hält also eine technische Infrastruktur vor, die ausschließlich dieser Form der Rückkopplung zwischen Anwendern und Entwicklern gewidmet ist. Schon ein erster Blick zeigt, daß es sich bei der Formulierung von Wünschen um ein häufiges Ereignis handelt: Bei einem Zugriff auf die Wishlist im Juli 2004 fanden sich auf dieser 817 Wünsche, die als nicht bearbeitet geführt wurden. Berücksichtigt man nun, daß ‚erfüllte' Wünsche aus der Liste ausgetragen werden, wird deutlich, wie stark die Anwender die Erreichbarkeit der Entwickler nutzen, um sich mit Ideen und Vorschlägen an sie zu wenden. Gleichzeitig zeigt die hohe Zahl der Einträge auch an, daß Wünsche wenig Aussicht auf Erfüllung haben: Sie konkurrieren mit mehreren Hunderten von anderen Anwendern geäußerten Vorschlägen um die Aufmerksamkeit der Entwickler, die daneben selbst noch eigene Ideen haben und diese verfolgen. Die Anwender versuchen also eine ‚voice'-Option wahrzunehmen, leiden aber darunter, daß sie meist kein Gehör

finden. Dieser Befund legt die Interpretation nahe, daß das System nur vorgehalten wird, damit die Wünsche der Anwender an einem Ort geäußert werden können, der die Kommunikation der Entwickler möglichst wenig ‚stört'.

Handelt es sich bei der Wishlist also um eine Einrichtung, welche primär das Ansehen des Projekts in der Öffentlichkeit der Benutzer fördern soll, oder finden sich unter den Wünschen interessante oder originelle Vorschläge, die von den Entwicklern für gut befunden und zum Entwicklungsziel gemacht werden? Unser Material gibt Anlaß zu einer zurückhaltenden Einschätzung. Weder finden sich in der Kommunikation auf der Mailingliste im Erhebungszeitraum Hinweise darauf, daß die Wünsche der Anwender den Zielfindungsprozeß maßgeblich beeinflußten, noch darauf, daß im Anschluß an die Wünsche Entwicklungsanstrengungen zur Implementation der entsprechenden Funktionalität unternommen wurden. Ein ähnliches Bild ergibt sich mit Blick auf die Interviews mit den Entwicklern von KMail. Auf die Frage nach dem Stellenwert von Wünschen der Anwender angesprochen, antwortet der Entwickler KD 1:

„Also, vieles sind so Wishlist-Sachen, an die man selbst schon gedacht hat. […] Und solche Wishlist-Items kommen dann auch. Das ist so ein Teil. Der andere Teil ist sind so Sachen, die sind teilweise relativ abstrus. Irgendwelche Filteractions, so Virenscanner aufrufen oder so was. […] Man tut sich immer schwer, Wishlist-Items zu schließen. Deshalb gibt's auch so viele davon, aber manchmal ist es dann wirklich so, daß man die dann auch schließt und sagt, ‚nee, das können wir niemals machen'. Aber viel von den Wishlist-Items, die da offen sind, das ist einfach so was, was zur Zeit keinen interessiert. Und da muß man halt drauf warten – oder die Leute, die das gepostet haben, mal warten –, bis mal einer sich dransetzt und das macht." (KD 1)

Chance auf Verwirklichung haben also nur diejenigen Wünsche, die der Entwickler selbst schon erwogen hat. Die Programmierer generieren ihre Ziele weitgehend eigenständig und berücksichtigen Wünsche nur in Fällen, bei dem sich ein Vorschlag mit dem vom Entwickler gewählten Ziel zufällig deckt. Allerdings wirft die schiere Existenz des Bug-Track-Systems und der darauf befindlichen Wünsche weitere Fragen auf: Warum hält ein Projekt eine solche Infrastruktur vor, wenn sie kaum genutzt wird? Warum verbleiben Wünsche im System – obwohl es den Core-Entwicklern durchaus bewußt ist, daß sich die Liste stetig vergrößert und unübersichtlich wird und obwohl das Löschen von Wünschen eine konsequenzlose low-cost-Aktivität ist? Tun sich die Entwickler, wie in der zitierten Passage angesprochen, mit dem Löschen von Wünschen schwer, weil sie vermuten, ein anderer Entwickler könnte den Wunsch

möglicherweise erfüllen? Im folgenden wollen wir eine andere Interpretation dieses Phänomens vorschlagen, da wir davon ausgehen, daß die länger beteiligten Entwickler recht gut einschätzen können, welche Wünsche im Verlauf des Projekts Chancen auf Realisierung besitzen und welche nicht. Um es vorwegzunehmen: Der ambivalente Umgang mit Wünschen – Verbleib von Wünschen auf der Liste trotz fehlender Chance auf Verwirklichung – ist Resultat einer positiv-wohlwollenden Einstellung gegenüber den Anwendern, die ihre Ursache in einem Gabenaustausch zwischen den beiden Akteursgruppen hat.

Um dieses Phänomen zu verstehen, ist es nötig, auf die Gründe für eine Beteiligung zurückzukommen. Hier haben wir festgestellt, daß Anerkennung einen stabilisierenden Faktor für Beteiligung darstellt und eine wichtige Voraussetzung dafür bildet, daß Programmierer ihre Entwicklungstätigkeit als sinnvoll interpretieren. Von Wert ist die Anerkennung insbesondere in Frühphasen, wenn sie noch keine regelmäßige Begleiterscheinung der Entwicklungstätigkeit darstellt. Die Weiterentwicklung des Programms einerseits und die Anerkennung der Entwicklungstätigkeit andererseits kann vor diesem Hintergrund als Austausch von sehr unterschiedlichen Gaben verstanden werden: Auf beides haben die Beteiligten keinen Anspruch, beides wird von ihnen geschätzt. Dieser Gabenaustausch führt dazu, daß die Anwender zumindest in einem gewissen Rahmen Einfluß auf den Entscheidungsprozeß über Ziele gewinnen. Zur Sprache kommt dieser Zusammenhang sehr deutlich in der folgenden Interviewpassage eines Entwicklers, der durch seine Autorentätigkeit für ein freies Software-Magazin über Einblicke in eine Vielzahl von Projekten verfügt:

> „Ich meine, die Leute[18] sind auch glücklich, daß ihr Projekt so geschätzt wird, die finden das auch toll, daß die Leute das mögen und […] [betont] ‚wenn ich das jetzt mache, dann kann das ja noch mehr, und noch besser und dann mögen das ja noch mehr Leute'. Das ist also dann einfach der Wunsch, noch besser zu werden, zu gucken, wo sich das Projekt hinentwickelt." (D 8)

Der Einfluß von Anwendern auf Entscheidungen über Zielsetzungen ergibt sich aus dem Streben nach einer Vergrößerung der Anerkennung. Damit das Programm für eine größere Anzahl von Anwendern attraktiv wird, orientieren sich die Entwickler an den von ihnen geäußerten Wünschen nach neuer Funktionalität. Dies kann zu einem höheren Maß an Rückkopplung der Anwender führen, die das Programm loben, für die

18 Gemeint sind hier die Entwickler von recht jungen Projekten, die sich von ihren Entwicklungszielen her stark an den Wünschen der Anwender orientieren.

Bereitstellung danken und einen sich verstärkenden Kreislauf des Austauschs von ‚Funktionalität' gegen ‚Anerkennung' in Gang setzen.

Wie vertragen sich diese Befunde mit den Einsichten, die wir an unserem Fallbeispiel KMail gewonnen haben? In unserem Material finden wir Hinweise darauf, daß Anerkennung durch Benutzer mit dem Erfolg eines Projekts einerseits zur permanenten Begleiterscheinung wird und andererseits das Hinzutreten von weiteren Entwicklern im Zuge einer Vergrößerung des Projekts Anerkennung in einem geringer werdenden Maße personalisiert.[19] Dadurch, daß Anerkennung mit der Verbreitung des Programms zu einer Selbstverständlichkeit wird, müssen keine besonderen Anstrengungen mehr betrieben werden, um in ihren Genuß zu kommen. Ähnliches gilt für die Personalisierung: Je stärker Lob und Dank an ein Kollektiv adressiert wird, in einem um so geringeren Maße fühlt sich ein individueller Entwickler davon angesprochen und im Gegenzug aufgefordert, die Wünsche von Anwendern zu Entwicklungszielen zu erheben. In diesen Kontext reiht sich auch der oben beschriebene, wenig rational anmutende Umgang mit der ‚Wishlist' ein: Die Zurückhaltung bei der Löschung von Wünschen resultiert aus dem durch den Gabentausch erzeugten ‚Commitment' der Entwickler gegenüber den Anwendern, das allerdings nicht als zwingend genug empfunden wird, als daß es zur Erwiderung der Gabe ‚Anerkennung' durch ‚Funktionalität' kommt.

Abschließend wollen wir in einem letzten Punkt darauf verweisen, daß der Einfluß von Benutzern auf Zielsetzungen nicht ausschließlich positive Effekte haben muß. Entgegen der naiven Vorstellung, der Einfluß führe zu einem höheren Maß an Anpassung und Benutzerfreundlichkeit, finden sich in unseren Interviews auch Anhaltspunkte dafür, daß dieser auch seine ‚Kosten' hat.[20] „Ganz häufig ist es so, daß sehr nützliche Programme irgendwann sehr unnütz werden, einfach deswegen, weil die Programmierer nie nein sagen konnten, und die Dinge dann irgendwann zu eierlegenden Wollmilchsäuen werden." (D 8) Problematisch kann Anwendereinfluß sein, wenn er zu Überfrachtung des

19 Auf der KMail-Mailingliste zeigte sich dies insbesondere darin, daß E-Mails von Anwendern, die an einzelne Entwickler persönlich adressiert sind und in denen den Entwicklern gedankt oder das Programm gelobt wird, von diesem auf die Mailingliste weitergeleitet werden.

20 Eine Beantwortung der Frage, ob der Einfluß von Anwendern größere Vor- als Nachteile hat, ist im Rahmen dieser Untersuchung nicht möglich. Sie würde einen systematischen Vergleich von möglichst ähnlichen Projekten erfordern, die sich aber hinsichtlich des Grades an Anwendereinfluß möglichst stark unterscheiden. Zu den Problemen einer Nutzerbeteiligung bei proprietärer Softwareentwicklung siehe Heinbokel 1994: 107 und Weltz/Ortmann 1992: 75 ff.

Programms mit Funktionalität führt. Da Anwender im Regelfall nicht einschätzen können, ob sich die gewünschte Funktionalität in sinnvoller Weise im Programmentwurf implementieren läßt, besteht die Gefahr, daß Ziele vorgeschlagen werden, die das Programmdesign überladen. Sofern die Entwickler sich verpflichtet fühlen, den Wünschen nahezu ausnahmslos nachzukommen, besteht die Gefahr der Verfolgung unangemessener Zielsetzungen. Zugespitzt würden Entscheidungen über Ziele nicht mehr auf der ‚Logik' argumentativer Prüfung basieren, sondern auf der ‚Logik' des Gabenaustauschs – mit den entsprechenden Folgen.

Zusammenfassung

In diesem Kapitel haben wir die Frage diskutiert, wie innerhalb von offenen Programmierprojekten über Zielsetzungen entschieden wird. Wir haben gesehen, daß an Entwicklungszielen prinzipiell kein Mangel besteht, sondern im Regelfall unterschiedliche Ziele gegeneinander konkurrieren. Offene Programmierprojekte zeichnen sich durch die Abwesenheit von Hierarchie oder eines anderen einheitlichen Entscheidungsmodus aus – die Kombination von unterschiedlichen Mechanismen bildet ein Strukturmerkmal der Projekte. Den basalen Modus bildet Argumentation: Entwicklungsziele müssen begründet und häufig gegen eine kritische Überprüfung verteidigt werden. Typische hier ins Feld geführte Begründungsnormen sind die Vergrößerung des Benutzerkreises im Sinne einer Popularisierung – häufig von Protagonisten zur Begründung des Ziels eingesetzt – und die Handlungsfreiheit des Anwenders. Argumentation zielt dabei auf die Herstellung von Einvernehmlichkeit, worunter nicht der Konsensus im Sinne einer Zustimmung aller, sondern die Abwesenheit von offenem Protest gemeint ist. Häufiges Ergebnis des Entscheidungsprozesses ist ein Kompromiß, der typischerweise den Einbau einer Konfigurationsoption beinhaltet, mit dem das Programm unterschiedlichen Bedürfnissen angepaßt werden kann. Ein weiterer Einflußfaktor in diesem Prozeß bildet Reputation. Dabei handelt es sich allerdings nicht um einen konkurrierenden Mechanismus der Herbeiführung von Entscheidungen, vielmehr sorgt Reputation dafür, daß eine Person zum relevanten anderen wird, mit dem auf argumentativen Wege Einverständnis hergestellt werden muß.

Argumentation bildet einen schwachen Modus der Herbeiführung von Entscheidungen: Differenzen – bedingt durch verschiedene Zielsetzungen, die mit unterschiedlichen Bewertungsnormen begründet werden – können die Weiterentwicklung blockieren. Hier greifen dann zwei Faktoren, die eine dauerhafte Blockade des Entwicklungsprozesses ver-

hindern. Zum einen ist dies die Ermüdung der Teilnehmer im Verlauf der Argumentation, die eintritt, da die Programmierer an der Entwicklungstätigkeit selbst, nicht aber in ähnlichem Maße am Diskussionsprozeß interessiert sind. Zum anderen ist das der Übergang zur Implementation der gewünschten Funktionalität in Fällen längerer Diskussionen über das Ziel. Die Beendigung von Entscheidungen durch Implementation ist nur dann legitim, wenn Entscheidung qua Argumentation bereits gescheitert ist.

In bezug auf die Rolle von Benutzern hatten wir gesehen, daß diese vom Entscheidungsprozeß über Zielsetzungen weitgehend ausgeschlossen bleiben: Exklusion findet dadurch statt, daß Anwender nicht oder nur sehr bedingt in der Lage sind, Argumente zu beurteilen oder selbst vorzubringen. Trotz dieser schwachen strategischen Position verfügen sie zumindest in jüngeren Projekten, in denen Anerkennung durch Benutzer ein recht seltenes Phänomen darstellt, über einen gewissen Einfluß auf die verfolgten Ziele. Dieser resultiert aus dem Gabenaustausch zwischen Entwicklern und Anwendern, in dem die Bereitstellung von Funktionalität durch Anerkennung belohnt wird. Die Anerkennung stellt dabei ein für Entwickler wertvolles Gut dar, bildet sie doch die Grundlage dafür, die Entwicklungstätigkeit als ‚sinnvoll' zu interpretieren. In Projekten, in denen Lob und Dank vergleichsweise seltene Ereignisse darstellen, kann das zu einer sich selbst verstärkende Spirale des Austauschs von Funktionalität und Anerkennung führen. Dieser Zyklus spielt nun allerdings in Projekten, in denen Anerkennung durch Anwender regelmäßige Begleiterscheinung der Entwicklungstätigkeit darstellt, keine Rolle, da dort eine Entwertung von Anerkennung durch ihr dauerndes Auftreten stattfindet. Zu beachten ist dabei, daß die Partizipation von Anwendern am Aushandlungsprozeß von Zielsetzungen nicht automatisch zu besser angepaßten Programmen führt. Ohne diese Frage abschließend klären zu können, haben wir in den Interviews Anhaltspunkte dafür gefunden, daß Nutzerpartizipation auch dysfunktionale Folgen haben kann, die in einer Überladung des Programmdesigns liegen. Hieraus folgt nun, daß zur Herbeiführung von Entscheidungen über Zielsetzungen die Logik des Gabenaustauschs (im Sinne der Berücksichtigung von Wünschen zur Erzielung von Anerkennung) der Logik der Argumentation (also die Prüfung und Begründung von Positionen unter kompetenten Peers) nicht zwangsläufig vorzuziehen ist.

Bugreports und Fehlerbeseitigung

Ein vielfach mit freier Software in Verbindung gebrachtes Merkmal ist ‚Stabilität', also die Robustheit des Programms in Anwendungssituationen und das Ausbleiben von fehlerhaften Reaktionen wie ‚Abstürzen'.[1] Wie ist diese Qualität zu erklären und welche Faktoren sind dafür verantwortlich? Die Standarderklärung, gegen die wir uns im folgenden Kapitel abgrenzen wollen, da wir sie für vereinfachend halten, lautet wie folgt: „Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone." (Raymond 1999: 41) Dieser Erklärung nach führt die Anwendung der Software durch eine große Anzahl von Personen dazu, daß Programmierfehler schnell aufgefunden und rasch beseitigt werden. Voraussetzung bildet die Offenheit des Quellcodes, der aufgrund dieser Eigenschaft evaluiert und korrigiert wird. So richtig diese Erklärung ist, so sehr unterschlägt sie die im Zuge des Prozesses von den Akteuren zu lösenden Probleme und greift daher an entscheidenden Stellen aus soziologischer Perspektive zu kurz: Die Erklärung unterstellt die eindeutige

1 Die Zuschreibung dieses Merkmals findet sich nicht nur in der Fachpresse (vgl. z.B. http://www.computerwoche.de/index.cfm?pageid=256&artid=33710&type=detail&category=67), sondern auch in Umfrageuntersuchungen. So kommt Wichmann (2002: 47) zu dem Schluß, daß an erster Stelle der Gründe für eine Nutzung von freier Software Stabilität und Schutz gegen den Zugriff auf Daten durch Unbefugte genannt werden. Auch in den sogenannten Halloween-Dokumenten, in denen ein Ingenieur von Microsoft die Gefahren der freien Softwareentwicklung für die Marktposition von Microsoft einschätzt, wird unter der Überschrift „OSS projects have achieved ‚commercial quality'" Robustheit und die weitgehende Abwesenheit von Fehlern hervorgehoben, die z.T. höher als die proprietärer Software sei (vgl. http://www.opensource.org/halloween/halloween1.php).

Identifizierbarkeit von Programmfehlern, eine problemlose Eingrenzbarkeit der Ursachen für das fehlerhafte Verhalten und behandelt die Stabilität als einen einfachen Größeneffekt.

Die soziologische Rekonstruktion des Vorgangs der Identifikation und Beseitigung von Bugs zeigt hingegen, daß es sich dabei keineswegs um einen trivialen Vorgang handelt: Die Kategorie ‚Bug' ist ebensowenig selbstverständlich wie der Schluß von einer Verhaltensweise eines Programms in einer Anwendungssituation auf fehlerhafte Algorithmen im Programmquellcode. Identifikation und Beseitigung von Programmfehlern setzen die Kooperation von Entwicklern und Anwendern voraus. Wie zu zeigen ist, ist die Stabilität freier Software Ergebnis rekursiver Lernprozesse, die durch realexperimentelle Implementationen in eine Vielzahl unterschiedlicher Anwendungskontexte initiiert und mit eher am klassischen Experiment orientierten Tests abgeschlossen werden.

Zum Aufbau dieses Kapitels: Erstens werden wir den Gegenstand des Kapitels durch die Klärung des Begriffs ‚Bug' bestimmen, um daran anschließend zweitens das begriffliche Werkzeug zur Erfassung des Prozesses der Fehlerbeseitigung – das Konzept ‚Realexperiment' – einführen. Drittens werden wir der Frage nachgehen, wie sich die Veröffentlichung und Anwendung von freier Software in diesen begrifflichen Rahmen einordnet. Hier werden wir argumentieren, daß die Veröffentlichung neuer Versionen zur Anwendung des Programms in unterschiedlichen Kontexten führt, wobei das verfolgte Ziel nicht nur in der Abarbeitung von Aufgaben im Anwendungsfeld, sondern – bezogen auf die Weiterentwicklung von Software – ebenso in der Erzeugung von Wissen über ‚Bugs' liegt. Nach dieser Einordnung analysieren wir viertens ein Fallbeispiel einer Fehlerbeseitigung auf der KMail-Mailingliste. Hier werden wir sehen, daß die Erzeugung des Wissens, das für die Beseitigung von Bugs notwendig ist, sich als Leistung weder dem Entwicklungs- noch dem Anwendungskontext zurechnen läßt, sondern gerade Resultat der Rückkopplung beider Kontexte ist. Die Kooperation basiert dabei auf institutionalisierten normativen Erwartungen, die wir in diesen Schritt mit analysieren werden. Das Kapitel schließt fünftens mit einer zusammenfassenden Verdichtung der Ergebnisse.

Definition ‚Bug'

Der Begriff ‚Bug' scheint auf den ersten Blick unproblematisch zu sein. Ein Bug oder Programmierfehler liegt dann vor, wenn in einer Anwendungssituation eine Prozedur oder das gesamte Programm nicht funktioniert. Eine ähnliche, an unser Alltagsverständnis angelehnte Definition

findet sich auch im „New Hackers Dictionary“: „**bug** *n*. An unwanted and unintended property of a program or piece of hardware, esp. one that causes it to malfunction. Antonym of feature.“ (Raymond 1998: 94) Eine solche Definition von ‚Bug‘ mag für proprietäre Programme gangbar sein, da die gewünschten Programmeigenschaften üblicherweise in Form einer Spezifikation oder eines Pflichtenhefts detailliert festgehalten werden, so daß Abweichungen des Programmverhaltens von den gewünschten Eigenschaften festgestellt werden können. Schwieriger ist bisweilen die Bestimmung eines Bugs im Fall von freier Software, da hier nicht vor Beginn der Entwicklung sämtliche Eigenschaften des Programms bekannt sind, sondern im Verlauf der Entwicklung ergänzt und verändert werden, ohne daß sich dies sofort auch in der Dokumentation des Programms niederschlagen muß. So ist es manchmal schwierig, ein Programmverhalten eindeutig als Fehler zu klassifizieren, wie die folgende E-Mail auf der KMail-Mailingliste zeigt:[2]

„How does KMail get the field ‚Sender‘ to display in the message list view? I looked at a few emails and for example – From: KU 1 [Name des Autoren, N.C.T.] is listed in the actual email but ‚KU 1‘ is in the Sender field.
Also when I send an email and put it into a folder (same topic) the from header has the person I sent the letter to and not mine.
Are these known problems?“ (KU 1; 02. Feb. 2000 03:21:24)

Der Autor schildert das Verhalten des Programms KMail beim Absenden einer E-Mail: Es trägt den Autoren als Absender einer E-Mail ein und zeigt dessen Namen in der Spalte ‚Sender‘ an. Dies verändert sich nun, sobald die E-Mail gesendet wird. Als Sender wird dann nicht mehr der Autor genannt, sondern der Empfänger. Von Interesse ist hier, daß der Anwender KU 1 das Verhalten als ‚Problem‘ interpretiert, da es nicht seinen Erwartungen entspricht. Offensichtlich geht er davon aus, der angezeigte Sender einer E-Mail solle immer derselbe bleiben, unabhängig davon, ob die Nachricht bereits versandt wurde oder nicht und in welchem Verzeichnis sie sich befindet.

2 Selbstverständlich gibt es auch im Bereich der freien Software eine Vielzahl von Programmreaktionen, die unproblematisch von jedermann eindeutig als ‚Fehler‘ erkannt werden können. Auch ist es prinzipiell möglich, durch einen Abgleich von Programmreaktionen und der Dokumentation Fehler zu identifizieren. Von Interesse sind für uns die Grenzfälle, da sich an ihnen zeigt, daß eine Programmreaktion nur dann als Fehler gilt, sofern sie von einem Entwickler als solcher klassifiziert wird. Die Verständigung auf eine gemeinsame Problemdefinition – bei der dann in aller Regel der Entwickler die Definitionshoheit hat – bildet einen wesentlichen Bestandteil des Prozesses der Fehlersuche und -beseitigung.

Die Antwort des Entwicklers zeigt nun, daß auch eine andere Interpretation möglich ist „This is a feature, not a bug. :-) Actually, the column in the message list, ‚Sender', should be named ‚Who' instead, as it might lead to misunderstandings." (KD 3; 03. Feb. 2000 10:25:05) Der Entwickler interpretiert das Programmverhalten hier als eine gewollte Funktionalität, räumt aber ein, die unterschiedlichen Interpretationen kämen durch eine unglückliche Bezeichnung einer Spalte zustande. Das Beispiel zeigt, daß die Charakterisierung eines Verhaltens als ‚Bug' nicht ‚objektiv' durch bestimmte Verhaltenscharakteristika zustande kommt, sondern von den Erwartungen an das Programm abhängig ist. Maßgeblich ist im Fall von unterschiedlichen Perspektiven die Interpretation der Entwickler. Von deren Beurteilung hängt es ab, ob eine Verhaltensweise als intendiert gilt oder als nicht-intendiert und fehlerhaft. Verhaltensweisen, die Erwartungen der Entwickler nicht entsprechen, setzen innerhalb von freien Softwareprojekten ein spezifisches Handlungsprogramm in Gang, das uns im folgenden interessieren wird.

Zum Begriff Realexperiment

Bevor wir uns dem Fall eines Programmierfehlers zuwenden, wollen wir im folgenden das begriffliche Werkzeug für die Analyse des Rückkopplungsprozesses zwischen Programmierern und Anwendern entwickeln, die zur Beseitigung von Programmfehlern führen. Wichtiges Kennzeichen von freier Software ist ihre ‚Unfertigkeit' bzw. die Unabgeschlossenheit der Entwicklung. Wie weiter oben dargestellt, bildet die jeweils aktuelle Version eines Programms immer nur einen vorläufigen Stand der Entwicklung, der durch neue Programmversionen überholt wird, die ihrerseits wiederum kein endgültiges Ergebnis sind.[3] Von Version zu Version wird neue Funktionalität entwickelt, implementiert und in Umlauf gebracht. Dies geschieht nicht nur vermittels des CV-Systems, sondern vor allen Dingen durch die Veröffentlichungen ‚offizieller Versionen', die bei unserem Fallbeispiel KMail an den Veröffentlichungszyklus des KDE-Verbundprojekts gekoppelt sind. Vermittels dieser Distributionswege gelangen neue Programmversionen in Umlauf, verlassen den Entwicklungskontext, in dem bereits eine erste Erprobung stattgefunden hat und gelangen in einen Einsatzkontext, in dem nicht nur kompetente Entwickler, sondern auch Anwender die Software zur Bewältigung von Aufgaben einsetzen. Wie ist die Veränderung von der Ent-

3 Damit soll allerdings nicht ausgeschlossen werden, daß die Entwicklung eines Programms zu einem Ende gebracht wird. Dieser Fall tritt ein, sofern sich niemand für die Weiterentwicklung eines Programms interessiert.

wicklung des Programms und dessen ersten erprobenden Einsatz hin zu einer Verwendung des Programms durch Anwender konzeptionell zu erfassen? Wir wollen vorschlagen, diese Überführung des Programms in seinen Anwendungsbereich als Realexperiment zu interpretieren, mittels dessen Wissen über das Verhalten des Programms in einer großen Anzahl von Verwendungskontexten gewonnen wird.

Unter Realexperimenten werden Groß/Hoffmann-Riem/Krohn folgend Eingriffe verstanden, die mit einem bestimmten Gestaltungsziel verbunden sind (2003: 242, 2005). Es grenzt sich vom Laborexperiment ab, da es nicht an einem separierten, auswirkungsgeschützten Raum stattfindet, sondern die Folgen des Experiments über die Grenzen des Labors hinaus spürbar sind. Durch das Fehlen eines Schutzes gegenüber Experimentalfolgen wird die ‚Gesellschaft zum Labor' (Krohn/Weyer 1989). Diese Entgrenzung des Experiments hat dem Konzept zufolge Konsequenzen für die Legitimation der Forschungstätigkeit: Während das klassische Experiment durch das Ziel der Gewinnung von Wissen legitimiert wird, reicht dies bei Realexperimenten nicht aus. Soziale Akzeptanz, sei es nun gegenüber spürbaren Folgen und/oder der Zumutung von Risiken, läßt sich nicht durch Wissensgewinnung erreichen, sondern nur durch das als erwünscht geltende Gestaltungsziel. Der experimentelle Charakter des Eingriffs wird daher verdeckt, das Realexperiment als die Implementierung von erprobtem Wissen deklariert (Krohn/Weyer 1989: 349). Die mit dem Konzept als Realexperimente interpretierten Phänomenbereiche reichen von Abfalldeponien (Krohn 1997b, 2003; Herbold 2000) über Seen (Hoffmann-Riem 2003) und Prärien (Groß 2003) bis hin zur Freisetzung gentechnisch veränderter Pflanzen (Krohn/van den Daele 1998), der Tschernobyl-Katastrophe und dem Atombombenabwurf über Hiroshima (Krohn/Weingart 1986, 1987).

Aufschlußreich ist die Verortung von Realexperimenten im Kontext anderer Formen von handelnden Eingriffen: Groß/Hoffmann-Riem/Krohn kreuztabellieren hierzu die Dimensionen ‚Wissen' mit der Ausprägung Wissenserzeugung und Wissensanwendung sowie die Dimension ‚Randbedingung', wobei sie hier zwischen kontrollierten und situationsspezifischen Randbedingungen unterscheiden.[4] In diesem Feld verorten die Autoren vier idealtypische Formen des Eingriffs (a) Laborexperiment, (b) Feldbeobachtung, (c) ökologische Implementierung und (d)

4 Die Wahl der Bezeichnung der Ausprägung ‚situationsspezifisch' und ‚kontrolliert' ist etwas unglücklich, da ‚Kontrolle' Merkmal einer Situation darstellen kann. Die Unterscheidung zwischen ‚kontrollierten' und ‚unkontrollierten' Randbedingungen wäre eindeutiger und würde den Anschluß an die wissenschaftstheoretische Diskussion zum Experiment vereinfachen.

technische Implementierung. Zwischen diesen existieren nun viele Übergänge, die in die Kategorie ‚Realexperiment' eingeordnet werden: Realexperimente bewegen sich damit im Spannungsfeld zwischen kontrollierten und situationsspezifischen Randbedingungen und tragen sowohl Merkmale der Wissenserzeugung als auch der Wissensanwendung.

Die Merkmale von Realexperimenten lassen sich wie folgt zusammenfassen: (a) Realexperimente zeichnen sich gegenüber Laborexperimenten durch weitreichendere Auswirkungen aus, die in der Tendenz die gesamte natürliche und gesellschaftliche Umwelt betreffen können. (b) Innerhalb von Realexperimenten wird sowohl Wissen erzeugt, als auch angewendet. (c) Die Kontrolle über das Experiment und dessen Folgen ist beschränkt. (d) Eine Legitimierung findet nicht durch Wissenserwerb, sondern durch das Gestaltungsziel statt.

Obwohl das Konzept ‚Realexperiment' auf den ersten Blick intuitiv einleuchtet und ein Rahmen zur Beschreibung von Wissensgewinnungsprozessen vor dem Hintergrund der empirisch zu beobachtenden Verlagerung von Forschungsaktivitäten aus dem Labor heraus in eine nicht abgegrenzte Umwelt hinein wünschenswert ist, greift es zu kurz. Unser erster Kritikpunkt am Konzept bezieht sich auf die Abgrenzung des Gegenstands. Wie oben gezeigt, heben die Autoren im Unterschied zum Laborexperiment das Vorhandensein eines Gestaltungsziels und die Folgenhaftigkeit des Eingriffs für die natürliche und gesellschaftliche Umwelt hervor. Mittels dieses Kriteriums gelingt zwar die Abgrenzung zum Laborexperiment, fraglich bleibt jedoch, wie es sich mit der Unterscheidung von anderem, außerhalb des Labors stattfindenden instrumentellen Handeln verhält. Im Fall des instrumentellen Handelns werden (sach-)technische Mittel zur Erreichung von bestimmten Handlungszielen eingesetzt und Erfahrungswissen erzeugt; ebenso übt der Handelnde nur eingeschränkt Kontrolle über den Handlungszusammenhang aus, sein Handeln kann durch Störfaktoren beeinflußt werden, so daß er sein Handlungsziel nicht erreicht und statt dessen nicht-intendierte Folgen mit hoher Reichweite verursacht. Eine Abgrenzung zu dem instrumentellen Handeln, das aufgrund von jeweils wechselnden, situationsspezifischen Bedingungen immer den Charakter des Versuchsweisen trägt, gelingt so nicht.[5]

Unser zweiter Kritikpunkt, der eng mit dem ersten zusammenhängt, betrifft die Grenzen von Realexperimenten, also den Gegenstandsbereich, den ein (soziologischer) Beobachter des Experiments zu analysie-

5 Dies wird von Groß (2003: 121) durchaus bemerkt, wenn er davon spricht, Realexperimente würden sich der Alltagsbedeutung von ‚Experiment' im Sinne eines Ausprobieren annähern.

ren hat.[6] Aufgrund des Fehlens eines handhabbaren Abgrenzungskriteriums zählt in der Tendenz die Totalität der (sozialen und natürlichen) Welt zum Gegenstandsbereich des Experiments – und zwar im ungünstigen Fall sogar auf unbestimmte Zeit. Für den Analytiker von Realexperimenten stellt sich damit erstens das praktische Problem, infinite Folgenketten beobachten und analysieren zu müssen – oder die Analyse abzubrechen und bestimmte Folgen-Folgen nicht weiter zu berücksichtigen.[7] Zweitens verwischt sich durch das Fehlen eines Abgrenzungskriteriums die Grenze zwischen Experiment und Beobachtung. Gerade im Fall von weitreichenden Handlungsfolgen stellt sich die Frage, ob noch sinnvoll von einem Experiment gesprochen werden kann, bei dem das Stattfinden eines Eingriffs betont wird, oder ob es sich – ab einem gewissen Punkt – um Experimentalfolgen handelt, die in der Folgenkette weitab von eigenem oder fremdem Eingreifen liegen und nur noch erlitten und beobachtet werden.[8]

---

6 Besonders deutlich zeigt sich dieses Problem in der Interpretation von „Tschernobyl als Experiment“ bei Krohn/Weingart (1987). Ausgangspunkt der Analyse bildet die Feststellung, die weltweit vorhandenen 300 Atomkraftwerke seien Unikate, so daß jedes sein eigener Testfall und der aussagekräftigste Test der Unfall sei (ebd. 1). Der ‚Radius‘ des Experiments schließt dann in der weiteren Analyse nicht nur die Sachtechnik des kernschmelzenden Kraftwerks und die Verseuchung der Landstriche mit radioaktivem Material, sondern auch die Glaubwürdigkeit völlig überfordert reagierender Institutionen (Behörden, Regierungen, Experten) der betroffenen Länder ein (ebd. 4 ff.). Zur Experimentalanordnung zählen hiernach nicht nur das Kraftwerk, die Nahrungskette und das politische System der damaligen Sowjetunion, sondern auch die unterschiedlichen Institutionen von benachbarten und von vom radioaktiven Niederschlag betroffenen Länder. Aufgrund des Fehlens eines handhabbaren Abgrenzungskriteriums zählt in der Tendenz die Totalität der Welt zum Gegenstandsbereich des Experiments und zwar auf unbestimmte Zeit. Hier ließe sich dann fragen, ob das Experiment bereits abgeschlossen ist, oder unser Institutionensystem – auch heute noch – Bestandteil des Versuchs darstellt.

7 Das Problem infiniter Folgen ist von der Technikfolgenabschätzung her bekannt. Dort wird der Anspruch nach einer umfassenden Abschätzung der Auswirkungen immer nur bedingt eingelöst, indem ab einem gewissen Punkt Folgen-Ketten abgeschnitten und nicht weiter berücksichtigt werden (Grunwald 2002: 72; Dierkes 1993: 79; Paschen/Petermann 1991: 33).

8 Hier sei wiederum auf die Interpretation von Tschernobyl als Experiment verwiesen. Durch den von den Autoren vollzogenen Wechsel der Systemreferenz (von der Beobachtung eines kernschmelzenden Kraftwerks zur Beobachtung der Kommunikation des Ereignisses in den Medien und der Reaktionen der verschiedenen Institutionen) stellt sich die Frage, wie eine Abgrenzung des Gegenstandsbereichs vorgenommen und begründet wird beziehungsweise wann die Analyse des Realexperiments abgebrochen werden kann. Zweitens verwischt sich durch die Behandlung unterschied-

Im folgenden wollen wir uns der neueren Diskussion zum Experimentalismus in der Wissenschaftstheorie zuwenden, um dort einige Aspekte zur Präzisierung des Begriffs ‚Experiment' aufzugreifen. Neben der Diskussion um die Rolle des Experiments innerhalb des Forschungsprozesses[9] bildet ein unstrittiger Bestandteil der Debatte die Betonung des artifiziellen Charakters von Experimenten, der sich insbesondere in der Hervorbringung von Phänomenen mittels des eingreifenden Handelns durch den Experimentator und dem Handlungsziel, der Gewinnung von neuem Wissen, zeigt (vgl. Hacking 1996: 259; Janich 1998: 101; Radder 2003a: 4). Des weiteren wollen wir aus dieser Diskussion die dort betonten normativen Orientierungen aufgreifen, denen experimentierende Forscher folgen: Dies ist einerseits das Streben nach maximaler Kontrolle über die Experimentalanordnung, also der unabhängigen Variable(n) sowie möglicher Störvariablen und die Vermeidung von Fehlern (Guala 2003: 1199), und andererseits die Orientierung an einem möglichst hohen Maß an Systematik des Vorgehens. Beide Charakteristika bedürfen für unsere Zwecke noch einer Präzisierung: Bei ‚Kontrolle über ein Experiment' handelt es sich um eine Variable, die unterschiedliche Grade kennt – und nicht nur die Pole ‚kontrolliert' und ‚unkontrolliert'. Dies läßt sich durch die Unterscheidung von unabhängiger Variable, abhängiger Variable und Störvariablen leicht nachvoll-

lichen Ereignisse (explodierendes Kernkraftwerk, verseuchte Landstriche, Behördenhandeln und verunsicherte Öffentlichkeit) als Bestandteile eines Experiments die Unterscheidung zwischen Experiment und Beobachtung. Während der Kernschmelze das experimentierende Handeln der Bedienmannschaft und damit ein Eingriff vorausgeht, stellen die Handlungen von Politiker und Behörden sowie der Öffentlichkeit Reaktionen auf von ihnen beobachtete und vorgefundene Umweltbedingungen dar. Anstatt die beschriebenen Ereignisse als Bestandteil eines Experiments zu interpretieren, wäre es mindestens ebenso plausibel, in bezug auf die die Bedingungen radioaktiver Verseuchung nicht herstellenden, sondern vorfindenden Akteure wie Politiker und Öffentlichkeit von Beobachtungen zu sprechen.

9 Zentraler Topos der Diskussion bildet die Auseinandersetzung mit der deduktiv-nomologischen Erkenntnistheorie Popperscher Prägung und der Rolle, die sie dem Experiment zuweist. Diese Rolle beschränkt sich bei Popper ausschließlich auf die Überprüfung von Theorien, nicht aber auf deren Generierung: Am Anfang des Forschungsprozesses steht Popper zufolge die Theorie, die singuläre Folgerungen deduziert, welche einem empirischen Test ausgesetzt werden (vgl. Popper 1982: 6, 1979: 427, 435). Das Falsifikationsprinzip weist dem Experiment also eine ausschließlich negative Rolle zu: Mit ihnen können Theorien verworfen werden, eine Generierung neuer Theorien vermittels von Experimenten ist nicht vorgesehen. Gegen diese Auffassung führt Hacking (1996: 258 ff.) ins Feld, das Verhältnis von Theorie und Experiment könne sich im Laufe des Forschungsprozesses wandeln, so daß es auch möglich ist, daß das Experiment der Theorie vorangeht.

ziehen. Damit es sich um ein Experiment handelt, muß der Experimentator die unabhängige Variable in jedem Fall kontrollieren.[10] Weiter ist es möglich, daß nur ein Teil der Störvariablen kontrolliert wird, ein anderer Teil hingegen nicht. In Fällen, in denen die vollständige Kontrolle von Störvariablen nicht gelingt, wird auch von Quasi-Experimenten gesprochen (Cook/Campbell 1979: 5 ff.). Mit unserem Kriterium ‚Kontrolle' wollen wir nun nicht darauf abheben, daß bei der Durchführung von Experimenten eine vollständige Kontrolle über die Störvariablen tatsächlich realisiert wird, sondern die Experimentatoren ein hohes Maß an Kontrolle anstreben, indem sie einen möglichst großen Bereich von Störvariablen auszuschließen versuchen. Das Kriterium ‚Orientierung an einem systematischen Vorgehen' hingegen meint, daß die Eingriffe nicht blind im Sinne von trial-and-error erfolgen, sondern daß der Experimentator die unabhängige Variable gemäß bestimmter Kriterien verändert. Ein solches Kriterium kann die Modifikation der unabhängigen Variable gemäß den Anforderungen eines Theorietests oder die systematische Hervorbringung eines Phänomens sein. Unsere bisherige Begriffsarbeit zum Experiment läßt sich wie folgt zusammenfassen: Konstitutiv für ein Experiment ist (a) der Eingriff, der (b) durch einen Experimentator erfolgt, (c) als wissenschaftliche Methode auf die Gewinnung von Wissen zielt und (d) sich an den wissenschaftlichen Normen von systematischem Vorgehen und möglichst weitgehender Kontrolle orientiert. Mittels dieser Kriterien sind wir in der Lage, Experimente als Unterkategorie des instrumentellen Handelns zu bestimmen und sie von anderen Formen zu unterscheiden.[11]

Wie grenzt sich nun das Realexperiment vom Laborexperiment ab? Im folgenden wollen wir Realexperimente als eine Unterkategorie von Experimenten bestimmen, verschieben das Abgrenzungsmerkmal im Vergleich zum eingangs referierten Begriff jedoch leicht, indem wir nicht die Reichweite der Folgen, sondern die Isolation der Experimentalanordnung von seiner Umwelt hervorheben. In Fällen, in denen ein geringes Maß an Isolation vorliegt, die Experimentalanordnung also durch ein hohes Maß an Eingebundenheit in die Umwelt und starke Kontextualisierung gekennzeichnet ist, wollen wir von Realexperimen-

10 Trifft dies nicht zu, kann ein Experimentator keinen Eingriff durchführen. Es handelt sich dann um eine Beobachtung, nicht um ein Experiment.

11 Distinktionsmerkmale gegenüber anderen Formen des instrumentellen Handelns sind hier das Handlungsziel der Gewinnung von Wissen und der Orientierung an den wissenschaftlichen Normen. Um zweifelsfrei zu bestimmen, ob ein Experiment vorliegt, müssen also die Handlungsziele des Akteurs und die normative Orientierung (methodische Orientierung) untersucht werden.

ten sprechen. Konsequenz der Umwelteingebundenheit ist, daß Wirkungsbeziehungen zur Umwelt zugelassen werden (müssen) und Experimente – wie im ursprünglichen Konzept beschrieben – weitreichende Folgen haben können. Während bei Realexperimenten eine Abgrenzung zwischen Experimentalanordnung und Umwelt nur schwer möglich ist, zeichnen sich Laborexperimente hingegen durch ein hohes Maß an Isolation und starke Dekontextualisierung aus. Hieraus resultiert eine geringe Reichweite der Folgen, die sich durch die räumliche Abgeschiedenheit auf das Innere des Labors beschränkt.[12] Der Grad der Kontextualisierung als Distinktionsmerkmal von Real- und Laborexperimenten schließt an die Ausführungen zum Grad der Kontrolle an: Realexperimente finden – da sie innerhalb eines bestimmten Kontextes durchgeführt werden – immer unter den Bedingungen eines geringeren Maßes an Kontrolle statt: Variablen aus dem Experimentalkontext können in nicht-intendierter und nicht-antizipierter Weise auf die beteiligten Variablen einwirken und die Phänomenerzeugung beeinflussen. Dies gilt nun gerade für das Laborexperiment nicht. Charakteristikum ist hier, daß die Separation von Experimentalanordnung und Störvariablen (nahezu) vollständig gelingt. Davon abhängig ist nun die Frage nach den Auswirkungen. Die Einbettung in einen nur teilweise oder kaum kontrollierten Kontext im Fall der Realexperimente ist dafür verantwortlich, daß das Experiment weitreichende Folgen haben kann – ein Umstand, der der Gesellschaft als Risiko zugemutet wird. Eine solche Zumutung existiert im Fall von Laborexperimenten nicht, oder nur in vernachlässigbarer Weise.

Offen ist bislang die Frage geblieben, wie das Realexperiment einzugrenzen und das Problem der infiniten Analyse von Folgenketten zu lösen ist. Wir schlagen im folgenden eine restriktive Grenzziehung vor, die den Vorteil besitzt, zwischen den Methoden ‚Experiment' und ‚Beobachtung' unterscheiden zu können, dies allerdings unter Inkaufnahme der Kosten, unsere Grenzziehung auf der Perspektive des Experimentators gründen zu müssen. Um die Eingrenzung zu entwickeln und damit die Bestandteile der Experimentalanordnung zu bestimmen, greifen wir wiederum einige Aspekte der Experimentalismus-Diskussion und der

12 In ähnlicher Weise unterscheidet Perrow technische Systeme hinsichtlich des Grads der Kopplung von Systemelementen. Eng gekoppelte Systeme zeichnen sich durch das direkte Einwirken von Systembestandteilen aufeinander aus, während lose gekoppelte Systeme Puffer besitzen, die ein manuelles Eingreifen zulassen (1984: 89 ff.). Dieser Terminologie nach würde es sich bei Laborexperimenten um weitgehend entkoppelte Systeme, im Fall von Realexperimenten um mehr oder minder eng gekoppelte Systeme handeln.

dort aufgeworfenen Frage nach dem Verhältnis von Theorie und Experiment innerhalb des Forschungsprozesses auf. Bereits eingangs haben wir bemerkt, daß die Rolle des Experiments dort nicht ausschließlich als Mittel zur Überprüfung von Theorien bestimmt wird, sondern ihm mittlerweile eine prominente Rolle bei der Generierung von Theorie zugeschrieben wird (Hacking 1983; Heidelberger 1998: 73; Radder 2003b: 162). Pointiert zugespitzt werden die unterschiedlichen Rollen von Experimenten im Forschungsprozeß von Steinle, der zwischen explorativen und theoriebestimmten Experimenten unterscheidet.[13] Während das explorative Experiment durch das in die Breite gehende Erforschen eines Phänomens mittels der Variation von Parametern gekennzeichnet ist (Steinle 1998: 281/282) und der Bildung von neuen Begriffen dient, meint Theoriegeleitetheit, daß Experimente im Rahmen einer Theorie geplant, entworfen, durchgeführt und ausgewertet werden (ebd. 286).[14] Dabei hebt der Autor hervor, daß die Variation von Bedingungen in den explorativen Phasen systematisch unternommen werden und daher kein ‚gedankenloses Herumspielen an Apparaten' darstellten.

Inwiefern tangiert nun die Unterscheidung von theoriegeleitetem und explorativem Experiment unsere Suche nach einem Abgrenzungskriterium für Realexperimente? Entscheidend ist, daß der Experimentator bei beiden Typen von Experimenten über bestimmte Vorstellungen des Wirklichkeitsbereichs verfügt, in den er eingreift. Während dies im Fall des theoriegeleiteten Experiments eine zu testende Theorie ist, mit der ein bestimmter Zusammenhang zwischen den Variablen postuliert wird, ist dies im Fall des explorativen Experiments zumindest das Wissen um ein Phänomen.[15] Obwohl sich die Vorstellungen über den Wirklichkeitsbereich im Fall des explorativen von denen des theoriegeleiteten Experimentierens unterscheiden, besteht ein gemeinsames Merkmal darin, daß die Experimentatoren über eine wie auch immer geartete Vorstellung ihres Gegenstands(-bereichs), in den sie eingreifen, verfügen. In Ermangelung eines besseren Begriffs wollen wir diese Vorstellungen im folgenden ‚Konzeptionen des Experimentators' nennen, die mindestens einen Gegenstands- oder Phänomenbereich bezeichnen und wenigstens eine qualitative Beschreibung des Bereichs beinhalten.[16]

---

13 Die Typologie basiert auf der Analyse der Laboraufzeichnungen von Ampère zum Elektromagnetismus.

14 Dieser Typus entspricht weitgehend der Bestimmung der Rolle von Experimenten bei Popper.

15 Vgl. für den Fall Ampère auch Steinle 1998: 281.

16 Um es am Beispiel von Tschernobyl zu verdeutlichen: Es wurde ein Versuch mit dem Reaktor unternommen – dies war der Weltausschnitt, auf den sich das Handeln der Bedienmannschaft bezog; die Auswirkungen auf die Umwelt und das Institutionensystem der betroffenen Länder wurden

Wir wollen im folgenden die ‚Konzeption des Experimentators' als Abgrenzungskriterium von Experimenten nutzen: Zum Experiment bzw. zur Experimentalanordnung gehören sämtliche in die Konzeption des Experimentators fallenden Gegenstände und Phänomene, die vom Experimentator eingesetzt, kontrolliert oder hervorgebracht werden und die er seinem eingreifenden Handeln zurechnet. Andere, aus der Konzeption des Gegenstandsbereichs herausfallende Komponenten oder Phänomene, die eventuell vom Experimentator registriert und die er ex post seinem Eingriff zurechnet, wollen wir als Folgen des Experiments bezeichnen.[17]

Wir fassen die Überlegungen zum ‚Realexperiment' zusammen: Um von Realexperimenten sprechen zu können, müssen die folgenden Bedingungen erfüllt sein: (a) Handlungsziel muß nicht nur die Gestaltung eines bestimmten Phänomenbereichs, sondern gleichermaßen die vom Experimentator angestrebte Gewinnung von Wissen sein. (b) Durch das Charakteristikum des Eingriffs und das Ziel der Gewinnung von Wissen qualifiziert es sich als einen spezifischen Typus von Experimenten. (c) Der eingreifende und Phänomen-produzierende Experimentator orientiert sich an den Normen eines möglichst systematischen Vorgehens und einem möglichst hohen Maß an Kontrolle über die Experimentalanordnung. Diese definitorischen Kriterien sind notwendig, um Realexperimente vom instrumentellen Handeln zu unterscheiden. (d) Um die Grenzen von Realexperimenten zu bestimmen, das Problem der Analyse infiniter Folgenketten zu vermeiden und Realexperiment von Feldbeobachtungen zu unterscheiden, schlagen wir vor, die Konzeption des Experimentators als Eingrenzungskriterium zu verwenden. Folgen, Ereignisse oder Konsequenzen von Experimenten, die aus der Gegenstandskonzeption herausfallen, können zwar beobachtet und vom Experimentator ex post seinem Handeln zugerechnet werden, stellen aber keinen Bestandteil des Experiments, sondern eine beobachtete Folge dar.

---

von verschiedenen Akteuren beobachtet – mit diesen Gegenständen wurde unserer Begriffsfassung nach nicht experimentiert.

17 Der für die Unterscheidung schwierigste Fall bildet die Produktion eines nicht vorhergesehenen Phänomens innerhalb eines Experiments. Da das Auftreten des Phänomens vom Experimentator weder intendiert noch antizipiert wird, handelt es sich unserer Abgrenzung nach um eine beobachtete Folge des Experiments, das zu einem späteren Zeitpunkt zum Gegenstand weiterer Experimente gemacht werden kann.

Einordnung: Der Release-Zyklus als Realexperiment

In diesem Abschnitt wollen wir zeigen, daß die Veröffentlichung und Anwendung von Programmversionen eine Praxis darstellt, die auf die Gewinnung von Wissen über Programmierfehler zielt, den oben entworfenen Kriterien genügt und daher als Realexperiment zu interpretieren ist. Dazu ist erstens zu zeigen, daß es einen handelnden Eingriff gibt, der auf die Gewinnung von Wissen zielt und an den Normen eines hohen Maßes an Kontrolle und hoher Systematik des Vorgehens orientiert ist. Darüber hinaus muß eine Gegenstandskonzeption angegeben werden können, auf den der Experimentator seinen handelnden Eingriff bezieht.

Im Fall von offenen Programmierprojekten übernehmen die Entwickler des Programms die Rolle von Experimentatoren: Indem sie ein Programm bereitstellen und es zur Anwendung empfehlen, sorgen sie für dessen Nutzung in einer Vielzahl unterschiedlicher Anwendungskontexte. Inwiefern handelt es sich nun dabei um einen Eingriff, der auf die Gewinnung von Wissen zielt? Evidenz für die These, mit der Veröffentlichung einer neuen Programmversion werde auch die Gewinnung von Wissen intendiert, findet sich in den Interviews, in denen die Entwickler die Fehlerhaftigkeit der von ihnen entwickelten Programme ansprechen und vor diesem Hintergrund die Erprobung des Programms in unterschiedlichen Anwendungsbereichen als Mittel zu deren Beseitigung thematisieren. Dies zeigt sich beispielsweise in der folgenden Passage, in der ein Programmierer die Rolle von Bugreports wie folgt einschätzt:[18]

> „Also, eine sehr wichtige Reaktion ist immer die Fehlermeldung. Also, wenn jemand sagt, das stürzt bei mir ab. Am besten dir auch noch sagen kann, warum. Also, das ist schon mal selten, aber wenn du die Information hast, es stürzt ab, das ist schon mal gut. Weil, dann kannst du Fehler suchen gehen. Das ist 'ne sehr, sehr wichtige Reaktion. Es wird auch fast immer unterschätzt, wie wichtig diese Reaktionen sind." (D 8)

Der Entwickler rechnet nicht damit, daß das von ihm entwickelte Programm fehlerfrei ist und betont die Relevanz und den Nutzen von Informationen, die ihm im Fall eines Bugreports von Benutzern übermittelt werden. Aus seiner Perspektive führt die Anwendung freier Software zur Erzeugung von Wissen über Probleme und Programmfehler, das ihn im günstigen Fall erreicht und als Ressource in die weitere Entwicklung

18 Dieser Entwickler hatte im Interview zuvor betont, Bugs würden eine unvermeidliche Begleiterscheinung der Softwareentwicklung darstellen.

einfließt. Die Erzeugung von Wissen über Bugs durch die Anwendung von Software infolge einer Veröffentlichung ist also antizipiert, intendiert und erwünscht. Unterstützt wird unsere Interpretation auch durch das vorgehaltene Bug-Track-System, mit dem die Informationen über Fehler gebündelt und der Informationsfluß vom Anwendungs- zum Entwicklungskontext strukturiert wird. Es bildet demnach einen wesentlichen Bestandteil der Experimentalanordnung, der zur Aufzeichnung der im Experiment gewonnenen Informationen dient.[19]

Wie verhält es sich nun mit dem Kriterium der Orientierung an einem möglichst hohen Maß an Systematik des Vorgehens und der Kontrolle? Da Entwickler die Anwender im Regelfall nicht kennen, können sie weder die Anwendungssituation kontrollieren noch überprüfen, ob Anwender in Fällen des Auftretens von Bugs diese auch tatsächlich dem Projekt melden. Allerdings zeigt die Kommunikation auf der KMail-Mailingliste, daß sich die Entwickler trotz der widrigen Rahmenbedingungen um eine systematische Erschließung des Wissens über Bugs bemühen. Deutlich zeigt sich dies im folgenden Beitrag eines Entwicklers, dem ein Bugreport eines Anwenders von KMail vorangeht: „Make your bug reports high quality ones by including: A description of your environment, KMail version, PGP type version, system info. A description of the problem. A description of how to repeat the problem. Provide a fix for the problem if one is known.“ (KD 4; 27 April 2000, 04:44:58) Hier expliziert der Entwickler KD 4 die normativen Erwartungen an einen Fehlerbericht von Anwendern, mit denen die Verteilung der Rollen innerhalb des Experiments deutlich wird. Die Entwickler des Programms sind zur Erschließung von Informationen auf die Kooperation der Anwender angewiesen, da sie die Durchführung des Experiments und die Gewinnung von Experimentaldaten selbst nicht kontrollieren können. Aufgrund dieses Defizits adressieren sie die normativen Erwartungen nach Systematik und Kontrolliertheit an die Anwender und versuchen so, diese zu ‚Komplizen‘ oder ‚Co-Experimentatoren‘ zu machen: Die Anwender sollen die Erhebung der Experimentaldaten übernehmen – und zwar möglichst systematisch.

Unsere Einordnung des Release-Zyklus freier Softwareentwicklung als Realexperiment läßt sich wie folgt zusammenfassen: (a) Die Entwickler des Programms bilden die Experimentatoren, die (b) durch die

19 Heidelberger unterscheidet zwischen Instrumenten, die primär (a) der Erfahrenserweiterung (produktiven Funktion); (b) der Phänomenstrukturierung (repräsentierende oder strukturierende Funktion); (c) der Beherrschung der Bedingungen (konstruierende und imitierende Funktion) dienen (1998: 80). Im Fall des Bug-Track-Systems handelt es sich um ein Instrument vom Typus b.

Veröffentlichung einer neuen Version das Programm aus seinem Entwicklungskontext entlassen und in den Anwendungskontext überführen. (c) Mit diesem Eingriff ist nicht nur intendiert, daß die Nutzer ihre Anwendungsprobleme lösen, vielmehr wird darüber hinausgehend die Gewinnung von Wissen über Programmfehler anvisiert. (d) Die Systematik und die Anleitung durch methodische Prinzipien zeigen sich in diesem Fall nicht im Vorgehen der Entwickler, sondern in den normativen Erwartungen, die an Anwender adressiert werden. Es wird erwartet, daß diese möglichst vollständig und systematisch über Fehler berichten, die wichtigsten Randbedingungen der Anwendungssituation nennen und damit das Wissen über den Fehler möglichst erschöpfend an das Projekt weitergeben. Ein weiterer Aspekt dieser Experimentiertätigkeit wird durch den Umstand deutlich, daß die Implementation des Programms in vielen Anwendungskontexten gleichzeitig stattfindet. Im Sinne unserer oben entwickelten Unterscheidung handelt es sich damit um Realexperimente mit stark explorativem Charakter, die der Hervorbringung von Phänomenen – also Bugs – dienen.[20]

Feinanalyse: ‚kmail 1.0.29.1 (and 1.0.28) crashes on this mailfolder'

Wie hat man sich nun den Prozeß des Auffindens und Beseitigens von Programmierfehlern im Fall von KMail im einzelnen vorzustellen? Die realexperimentelle Erprobung freier Software in privaten oder beruflichen Anwendungssituationen selbst entzieht sich unserer Beobachtung, nicht aber der Kooperationsprozeß zwischen Entwicklern und Anwendern. Dieser beginnt üblicherweise mit einem Fehlerbericht eines Anwenders, der entweder über das Bug-Track-System oder die Projekt-Mailingliste dem Projekt übermittelt wird. Ein typischer Fall eines Bugreports bildet der Beitrag des Anwenders KU 2, der zur Übermittlung des Reports unter dem Subject ‚kmail 1.0.29.1 (and 1.0.28) crashes on this mailfolder' die folgende E-Mail direkt an die Mailingliste richtet:

20 Analoge Vorstellungen über den Transfer eines Programms von einem separierten Entwicklungskontext hinein in einen nicht-kontrollierten Anwendungskontext finden sich für die proprietäre Softwareentwicklung auch in der Literatur zum Software Engineering. Hierzu führen Endres/Rombacher (2003: 25) aus: „Testing is the development step that moves a system from the drawing board (albeit an electronic one) into operation, placing the product close to ist production environment. It usually begins in a controlled laboratory environment and gradually moves out into the actual user environment."

„I attached a file which at least for me crashes kmail when attempting to open it. It consist of only one message (a digest of the mandrake-expert list). I have already shortened that one message as far as possible. Further shortening results in the folder opening up correctly. Sorry for not tackling this problem myself, but I'm not yet in the KDE-developing (e.g. I don't even know how to start the debugger). […] P.S. The same has happend before, always with this same mailinglist (mandrake-expert-digest).“ (KU 2; 29. März 2000, 09:36:03)

Der Anwender beschreibt in der zitierten E-Mail ein Verhalten des Programms KMail, das offensichtlich den Erwartungen nach ordentlicher Funktionsfähigkeit nicht entspricht: Den Absturz des Programms beim Versuch, eine bestimmte E-Mail zu öffnen. Dabei teilt er einige Kontextinformationen mit, mit denen er die Bedingungen des Auftretens beschreibt und von denen er sich erhofft, sie könnten Hinweise auf die Ursache geben. Der Ertrag dieser ersten, realexperimentellen Erprobung des Programms besteht in der erfolgreichen Hervorbringung des Phänomens ‚nicht erwartungsgemäße Programmreaktion‘ in einer spezifischen Situation. Die Erzeugung des Phänomens ist dabei an die beim Anwender vorliegenden Bedingungen gebunden, da die Ursachen für den Fehler zu diesem Zeitpunkt noch nicht isoliert wurden und daher noch nicht bekannt sind. Im obigen Fall hat sich der Anwender bereits bemüht, die Fehlerursache einzugrenzen. Konkret heißt das hier: Die Kontextbedingungen zu verändern, den für die Fehlererzeugung relevanten Kontext zu verkleinern und die Arbeit zur Identifikation der Ursachen für die Entwickler zu verringern. Diese Tätigkeit kann bereits als phänomenerkundendes Experimentieren bezeichnet werden, da der Anwender mit der Variation des Kontextes – Verkürzung der E-Mail und damit Verringerung der in Frage kommenden Ursachen – die Bedingungen eines Auftretens zu eruieren beginnt. Bereits eine knappe Stunde nach dem Bugreport reagiert der Entwickler KD 5 auf den Beitrag und berichtet:

„I can open your attachment without problems (KMail 1.0.28). I inserted it into my inbox and kmail started ok. So I guess something is wrong with your mailboxes. Are you auto-compacting them? (File->Settings, tab ‚Misc‘, check ‚Compact all folders on exit‘) I'd suggest to backup your ~/Mail dir first!“ (KD 5, 29. März 2000, 10:30:03)

Offensichtlich hat dieser Entwickler das vom Anwender zur Verfügung gestellte Material genutzt und versucht, einen Absturz herbeizuführen.[21]

21 Der Versuch der Reproduktion des Phänomens ist recht üblich, wie die Reaktion des Entwicklers KD 3 zeigt. Dieser vermeldet, er habe den Fehler ebenfalls nicht erzeugen können (KD 3, 29. März 2000, 10:45:16).

Hierbei handelt es sich – ebenso wie bei der Erprobung des Programms durch den Anwender – um ein Experiment, allerdings mit anderem Charakter. Im Unterschied zur realexperimentellen Erprobung des Programms dient es nicht dem Auffinden eines Phänomens, sondern dessen Erkundung. Auch der Kontext des Experiments verändert sich hier: Es findet nicht im Rahmen einer Anwendungssituation statt, in der externe Ziele verfolgt werden, sondern als eine Praxis, die ausschließlich auf den Erwerb von Wissen über das Phänomen ausgerichtet ist. Da der Entwickler mit dem Auftreten eines Absturzes rechnet, wird er vermutlich Vorkehrungen gegenüber möglichem Datenverlust – wie z.B. der Durchführung des Experiments auf einer ‚Entwicklungsmaschine'[22]– getroffen haben und das Experiment mithilfe von Entwicklungswerkzeugen kontrollieren.[23] Von seinen Rahmenbedingungen her nähert sich der Test also dem Laborexperiment an.

Ein Erfolg bleibt jedoch aus, die E-Mail öffnet sich auf dem Computer des Entwicklers in vom Programm vorgesehener Weise. Welche Frage liegt diesem Experiment nun zugrunde? Es ist die Hypothese, ‚Die vorliegende Textmail führt aufgrund eines Programmfehlers zu einem Absturz von KMail', die nun allerdings durch das Experiment falsifiziert wurde. Dies erweist sich als folgenhaft für das weitere Vorgehen, wie der letzte Satz der E-Mail zeigt: Der Fehler wird nicht mehr dem Programm zugerechnet, sondern den Daten in der Mailbox in Kombination mit den Benutzereinstellungen. Der Test führt also zur Falsifikation der einen und zur Formulierung einer anderen Hypothese über die Ursachen des Absturzes. Da es sich um ein lokal auftretendes Phänomen handelt und dessen Ursachen in lokalen Gegebenheiten vermutet werden, verlagert sich die Experimentiertätigkeit wieder hin zum Anwender, der kurze Zeit später den folgenden Bericht abgibt.

„Well, it does happen to me when using 1.0.28 – confirmed. One thing I noticed just there: the first time I tried to open it, it sort of worked, though the index was sort of messed up. The reason was, I had an old .buggy-folder.index lying around. After removing this old file, it sure enough crashed again. One more thing I tried, is to make an empty mail folder with only the buggy-folder in it. Doesn`t help either. Yes, ‚compact all folders on exit' is active. Could you point out to me how I can trace that problem any better? Any other suggestions appart from the ‚compact all folders on exit' thing? Maybe you want to recommend some start-up reading for debugging a KDE-app?" (KU, 2 29. März 2000, 11:43:31)

22 Ein Computer, der ausschließlich zum Zweck der Programmentwicklung betrieben wird.

23 Z.B. mittels des noch weiter unten auftauchenden Debuggers.

Der Anwender macht sich zum Co-Experimentator, indem er dem vorgeschlagenen Vorgehen folgt und systematisch eine Fehlerursache nach der anderen ausschließt. Der Programmabsturz erweist sich allerdings als hartnäckig, er tritt weiterhin auf. Auch diese Hypothesen scheitern, da sie durch die Experimente falsifiziert werden. An diesem Punkt wendet sich der Anwender mit zwei Fragen erneut an die Entwickler: Zum einen erkundigt er sich nach Möglichkeiten, den Fehler besser nachverfolgen zu können. Mit dieser Frage möchte der Anwender also die Informationen über den Kontext, in dem der Fehler entsteht, systematischer erheben. Diese verweist auf die Notwendigkeit von Informationen, die die Grundlage für die Formulierung von neuen Hypothesen über die Fehlerursache bilden. Zum anderen fragt der Anwender die Entwickler nach weiteren Ideen zur Ursache für den Absturz. Dies zeigt, daß Anwender nur äußerst eingeschränkt über die Kompetenz zur Formulierung von Hypothesen verfügen, die dann im Fortgang geprüft werden könnten.

„Well general procedure is this: Open kmail in debugger, typing ‚gdb kmail'. Type ‚r' to start it. When it crashes, type ‚bt' to see backtrace. For this to work, you should have kmail, kdelibs and qt compilled with debug options on. Without all or some of them having debug info, backtrace won't show much info." (KD 5, 29. März 2000, 12:37:35)

Der Entwickler erläutert hier knapp die Funktionsweise des ‚Debugger'.[24] Mit ihm kann ein Programm ‚protokolliert' ausgeführt werden, so daß im Fall eines Absturzes die Zeile im Quellcode angegeben wird, an der er stattgefunden hat. Die Experimentalanordnung beim Anwender wird hier also ergänzt, wobei allerdings die durch das Programm produzierte Information über den Verlauf des Experiments nur von Entwicklern zu interpretieren ist. Die Formulierung von Hypothesen setzt voraus, daß die Programmanweisungen gelesen und verstanden werden.[25] Unsere Episode des gemeldeten und nachverfolgten Fehlers endet nun nicht mit dem Verweis des Anwenders KU 2 auf die fehlerhafte Zeile des Programmquellcodes, die er mit Hilfe des Debuggers ermittelt hat, an die sich dann die Beseitigung des Fehlers durch einen Entwickler anschließen könnte, sondern mit folgendem Beitrag eines hoch engagierten

24 Die Erläuterung bezieht sich auf den im Projekt KMail üblicherweise eingesetzten ‚GNU Project Debugger', ebenfalls freie Software. Siehe http://www.gnu.org /software/gdb/gdb.html (Stand 06/2004).

25 In der Literatur zum Software Engineering werden die unterschiedlichen Kompetenzen bei der Interpretation von fehlerhaften Programmen wie folgt thematisiert: „Users don't observe errors or faults. They observe execution failures." (Mills zit. nach Endres/Rombach 2003: 123).

Entwicklers: „It also works fine for me with KMail 1.0.29.1 (at the office have to get back to work now)." (KD 4, 30. März 2000, 5:42:16) Dieser hat im Zuge der Anwendung der neusten Version ebenfalls keine fehlerhafte Programmreaktion feststellen können und läßt auch keine Bereitschaft erkennen, dem Problem weiter nachzugehen. Vermutlich geht er davon aus, daß der Fehler in der aktuellen Version nicht mehr existiert oder hält die Aussicht auf Identifikation der Ursache für gering. Interessant ist für uns, daß dieser Beitrag die Anstrengungen um eine weitere Nachverfolgung des Fehlers beendet. Dies bildet ein Indiz dafür, daß hoch engagierte Entwickler auch im Bereich der Verfolgung und Beseitigung von Fehlern eine Leit- und Orientierungsrolle innehaben, die sich auf ihre Reputation gründet.

Zusammenfassung: Die ‚Stabilitäts'-Rekursion

In diesem abschließenden Abschnitt wollen wir die Argumentation des Kapitels zusammenfassen und zu einem Modell verdichten. Ausgangspunkt bildete für uns die Frage, wie die hohe Stabilität von freier Software zu erklären ist. Sie resultiert aus einer Rückkopplung von Anwendungs- und Entwicklungskontext, mit der Wissen über Programmierfehler erzeugt wird, das die Grundlage zur Beseitigung dieser Fehler bildet.

Dieser Rückkopplungsprozeß setzt sich aus mehreren Komponenten zusammen: Er beginnt mit der Veröffentlichung von neuen Programmversionen, mit der die Software von einer weitgehend dekontextualisierten Entwicklungsumgebung in eine Vielzahl unterschiedlicher Anwendungskontexte überführt und dort erprobt wird. Da sich die Kontexte in bezug auf die sachtechnische Umgebung wie Hardware, Betriebssystem und andere Applikationsprogramme stark unterscheiden, die Nutzer mit der Anwendung des Programms verschiedene Zwecksetzungen verfolgen und unterschiedliche Arten von Daten mit dem Programm bearbeitet werden, handelt es sich hierbei um eine realexperimentelle Erprobung des Programms, die auch der Gewinnung von Wissen über Programmierfehler dient. Durch die gleichzeitige und vielfältige Kontextualisierung des Programms wird schnell ein hohes Maß an empirisch-praktischem Wissen[26] im Umgang mit dem Programm erzeugt. Entwickler werden so zu Experimentatoren, sind aber, was die systematische Aufzeichnung des Wissens angeht, auf die Kooperation mit Anwendern angewiesen. Dies zeigt sich unmittelbar in den normativen Erwartungen,

26 Im Sinne des Konzepts der Innovationsnetzwerke. Vgl. hierzu Asdonk/Bredeweg/Kowol 1991, 1994.

die Entwickler an die Fehlerberichte der Anwender stellen: In ihnen sollen die Informationen über die Bedingungen des Auftretens von Bugs möglichst vollständig und erschöpfend dargestellt werden, um ein weiteres phänomenerkundendes Experimentieren zu ermöglichen. Bei der Erprobung der Software handelt es sich zusammenfassend gesprochen um ein Realexperiment mit stark explorativem Charakter, da in der Regel keine Annahmen darüber existieren, ob Fehler vorhanden sind und wie sich diese in der Anwendung äußern.

Zweitens werden die gewonnenen Experimentaldaten – das Wissen über Bugs – vom Anwendungskontext in den Entwicklungskontext mittels Bug-Track-System oder Entwickler-Mailingliste transferiert und zur Konstruktion weiterer phänomenerkundender Experimente eingesetzt. Dabei handelt es sich dann nicht mehr um Realexperimente, da die Kontexte, die eine Reproduktion des Fehlers stören könnten, nach Möglichkeit ausgeschaltet werden, sondern um eine Form, die sich von den Rahmenbedingungen her in der Tendenz dem Laborexperiment annähert. Wie wir gesehen haben, kann die Experimentiertätigkeit vom Entwicklerkontext wiederum in den Anwendungskontext zurückverlagert werden, sofern fehlerverursachende Faktoren nur ausschließlich dort vermutet werden. An unserem Beispiel hatten wir gesehen, daß die Beteiligten dann Anstrengungen unternehmen, mit einer anspruchsvolleren Experimentalapparatur (wie einem Debugger) die Aufzeichnung der Daten zu verbessern und so den Schluß von einem unerwünschten Programmverhalten auf einen fehlerhaften Bestandteil des Quellcodes durchführen zu können: Eine Aufgabe, für deren Bewältigung die betreffende Person über die Kompetenz verfügen muß, den Quellcode zu lesen und die damit nur von Entwicklern geleistet werden kann.

Das Merkmal ‚Stabilität' bildet im Fall freier Software somit Resultat aus der Zusammenführung von empirisch-praktischen (Wissen um Fehler) und theoretisch-systematischen Wissen (Wissen um Zusammenhänge von Faktoren innerhalb des Programms, Wissen über Regeln der Formulierung von Hypothesen, Kompetenz zur Interpretation der Experimentaldaten), das innerhalb von experimentellen Praktiken in kaskadenartigen Lernprozessen von (1) Implementation neuer Funktionen, (2) realexperimenteller Erprobung und (3) laborexperimentartiger Fehlerdetektion erzielt wird.

Zusammenfassung: Die soziale Konstruktion freier Software

In den vorangegangenen drei Kapiteln war die Frage leitend, wie eine Entwicklung von freier Software unter auf den ersten Blick sehr ungünstig erscheinenden Umständen möglich ist. Wie gelingt es den Entwicklern von freier Software unter den Bedingungen der Abwesenheit von Hierarchie und einer klaren Rollenstruktur, einer Projekt-Mailingliste als Medium der Kommunikation sowie einer hohen Fluktuation der Teilnehmerschaft, ihr Handeln auf ein gemeinsames Entwicklungsziel hin zu koordinieren? Diese recht umfassende Fragestellung haben wir zerlegt und verschiedene Facetten des Problems der Handlungskoordination diskutiert: Dazu gehörte die Frage nach den Problemen und typischen Lösungen von Projektgründungen, die Frage nach dem Zustandekommen von gemeinsam verfolgten Zielsetzungen sowie die nach den Mechanismen der Beseitigung von Programmierfehlern. Dabei sind wir nicht bei der Analyse der Mechanismen stehengeblieben, die eine Lösung dieser Anforderungen ermöglichen, sondern haben darüber hinausgehend auch das Ergebnis des Innovationshandelns – die Gestalt freier Software – im Auge behalten.

Die Organisation von Innovation als soziale Gestaltung von Technik

Im folgenden, diese Überlegungen abschließenden Schritt wollen wir die Ergebnisse der Untersuchung zur Handlungskoordination nutzen, um einen Brückenschlag zwischen zwei Richtungen der Innovationsforschung

zu versuchen: zwischen Konzepten, die sich primär für die soziale Organisation der Innovation interessieren – wie z.B. der Ansatz der Innovationsnetzwerke –, und solchen, die sich auf die soziale Gestaltung von Technik beziehen. Unsere These ist dabei, daß die organisationalen Rahmenbedingungen, unter denen der Prozeß der Technikentwicklung stattfindet, Faktoren darstellen, die Einfluß auf die soziale Gestalt von Technik nehmen. Es geht also um die Gestaltungswirkung der Praxis der Technikentwicklung auf das produzierte Artefakt, die wir anhand von freier Software nachzeichnen wollen. Evidenz für diese These findet sich bei der Analyse der Gründung von freien Softwareentwicklungsprojekten, der Entscheidung über Zielsetzungen und der Beseitigung von Programmierfehlern.

In bezug auf die ‚Stunde Null' freier Softwareentwicklungsprojekte haben wir gesehen, daß sich hier widersprüchliche Anforderungen stellen: Der Entwurf hat einerseits kohärent zu sein, um ein arbeitsteiliges Vorgehen zu gestatten, muß aber andererseits flexibel und offen gegenüber weiteren Zielsetzungen sein, um eine Integration neu hinzukommender, an bestimmten Entwicklungsproblemen interessierter Programmierer zu erlauben. Modularisierung stellt in diesem Zusammenhang eine Antwort auf diese Anforderungen dar, mit der beide Ziele zu einem gewissen Maß erreicht werden können. Die Strukturierung des Programms in Module – als Eigenschaft des Artefakts – resultiert im Fall von freier Software nicht nur aus der Komplexität der zu lösenden Entwicklungsaufgabe, sondern auch aus den Integrationserfordernissen, die sich aufgrund der motivationalen Gründe für eine Beteiligung auf Seiten der Entwickler und der organisatorischen Rahmenbedingungen im Fall von offenen Programmierprojekten stellen.

Bei der Analyse von Entscheidungsprozessen über Zielsetzungen hat sich gezeigt, daß die Argumentation den grundlegenden Modus bildet. Es werden unterschiedliche Entwicklungsziele vorgeschlagen und Argumente in der Diskussion ausgetauscht. Dabei gilt eine prinzipielle Begründungspflicht für die jeweilige vertretene Position – insgesamt ein schwacher Modus der Entscheidungsfindung, da aufgrund divergierender Bewertungsnormen die Möglichkeit eines hartnäckigen Dissens besteht. Daneben stellt der Einbau von Konfigurationsoptionen einen häufig anzutreffenden Kompromiß dar und die Protagonisten von Entwicklungszielen gehen bisweilen von der Diskussion zur Implementation über, um Blockadesituationen zu lösen. Weiter wurde deutlich, daß Anwender lediglich aufgrund des Gabenaustauschs – Anerkennung gegen Funktionalität – Einfluß auf die innerhalb des Projekts verfolgten Zielsetzungen gewinnen, der unter Umständen zu einer Überladung des Programmentwurfs führen kann. Auch hier sehen wir, daß die Mechanis-

men der Entwicklungspraxis die Gestalt freier Software prägen: Die weitgehende Konfigurierbarkeit und der Reichtum an Funktionalität im Fall von reiferen Programmen resultiert nicht zuletzt aus der großen Anzahl von sich meist sporadisch beteiligenden Entwicklern, die verschiedene Zielsetzungen verfolgen, jedoch nicht mit jedem vorgeschlagenen Ziel einverstanden sind. Die Offenheit der Projekte gegenüber neu hinzukommenden Programmierern einerseits und die Möglichkeit für Anwender, sich mit Vorschlägen direkt an die Entwickler zu wenden, andererseits bilden hier Faktoren, die Spuren in der Gestalt von Software hinterlassen.

Die Beseitigung von Fehlern und die damit verbundene experimentelle Praxis einer Überführung des Programms in eine Vielzahl unterschiedlicher Anwendungskontexte bei anschließender laborexperimentartiger Fehlerdetektion bilden integrale Bestandteile der Entwicklung von freier Software, auf die die hohe Stabilität zurückzuführen ist. Bugs werden im Unterschied zur proprietären Softwareentwicklung nicht ausschließlich vor einer Veröffentlichung von den Entwicklern gesucht, vielmehr bilden die häufigen Releases im Bereich der freien Software die Voraussetzung für eine Anwendung der Programme, in deren Zuge Anwender Wissen über unerwünschte Programmreaktionen sammeln. Die Organisation der Entwicklung gliedert sich dabei in schnell aufeinander folgende Release-Zyklen, in denen Anwendungs- und Entwicklungkontexte miteinander vielfach über Mailingliste und Bug-Track-System kurzgeschlossen werden.

Institutionalistischer Ursprung der Handlungskoordination

Geht man nun der Frage nach, welche Ursachen die Rahmenbedingungen der kooperativen Programmentwicklung in offenen Programmierprojekten haben und auf welchen Ursprung die handlungskoordinierenden Mechanismen zurückgehen, so wird deutlich, daß offene Programmierprojekte weitgehend institutionell bestimmt sind. Diese institutionelle Rahmung hatten wir bereits bei der Erklärung der Entstehung freier Softwareentwicklung durch die vertragliche Innovation freier Softwarelizenzen einerseits und die organisatorische Innovation offener Programmierprojekte andererseits analysiert. Deutlich wurde, daß die Gabenökonomie, die auf der Verpflichtung zur Gegengabe basiert – das Ethos freier Softwareentwickler, das sich aus den Bestandteilen Universalismus, Kommunismus, Skeptizismus und Uninteressiertheit zusammensetzt, hier den relevanten normativen Orientierungshorizont abge-

ben, vor dessen Hintergrund die Entstehung dieser Form der Softwareentwicklung verstanden werden muß. Der institutionalisierte normative Horizont wirkt in vielfältiger Weise auf die Softwareentwicklung ein. Die Anforderungen nach einer Kohärenz und Offenheit des ersten Programmentwurfs während der Gründung des Projekts erweist sich als institutionell vorstrukturiert: Die Verpflichtung zur Gegengabe führt zur Gründung von Projekten, das Streben nach Anerkennung unter Peers zu Bemühungen, das Projekt personell zu vergrößern – ein Ziel, das jedoch nur dann zu realisieren ist, sofern instrinsisch interessierten Programmierern der Platz eingeräumt wird, sich ‚eigensinnig' an der Entwicklung zu beteiligen und selbstgesetzte Zielsetzungen zu verfolgen. Die Modularisierung bildet dementsprechend eine Lösung der durch den institutionellen Rahmen vorgegebenen Anforderungen.

Als ähnlich einflußreich erweisen sich die institutionalisierten Orientierungen im Fall von Zielfindungsprozessen. Hier hatte es sich gezeigt, daß Kompetenz eine wichtige Voraussetzung bildet, um sich an den entscheidungsherbeiführenden Diskussionen zu beteiligen. Die Beteiligten orientieren sich deutlich an der Norm des Skeptizismus, wenn sie Argumente für einen Vorschlag vorbringen, mögliche Einwände antizipieren und im Voraus begegnen, Vor- und Nachteile erwägen und auf dieser Basis Entscheidungen treffen. Selbst dort, wo eine solche Orientierung in den Hintergrund tritt – namentlich bei der Berücksichtigung der Wünsche von Anwendern – sind Entscheidungen über Zielsetzungen durch Institutionen beeinflußt. Maßgebliche Orientierung bildet dann nicht das Ethos freier Softwareentwickler und die Norm Skeptizismus, sondern die Norm der Verpflichtung zur Gegengabe einer Gabenökonomie, in der Funktionalität gegen Anerkennung ausgetauscht wird.

Wir fassen diese Überlegungen zur sozialen Gestaltung von Technik innerhalb von offenen Programmierprojekten zusammen: Um die Merkmale freier Software zu erklären, ist es notwendig, neben anderen Faktoren auch die Praxis der Softwareentwicklung mit den dort vorfindbaren Problemen der Handlungskoordination und den dafür entwickelten Lösungen zu berücksichtigen. Sie wirken in vielfacher Weise auf den Entwicklungsprozeß ein und erklären zumindest zum Teil die Gestalt der Technik. Neben dem mittlerweile zum Standardrepertoire der Technikgenese- bzw. Innovationsforschung zählenden Faktoren wie Akteursinteressen, Problemdefinitionen, Systemrationalitäten, technische Stile, Konstruktionstraditionen und Paradigmen fordert die hier entwickelten Perspektive die Berücksichtigung der Erfordernisse der praktischen Entwicklungstätigkeit zur Erklärung der Gestalt von Technik. Über diese Brücke gelingt es, von Konzepten der Organisation von Innovationen zu Fragen nach den technikgestaltenden Faktoren zu gelangen.

Die Erfordernisse von Innovationshandeln entstehen jedoch nicht auf einer Tabula rasa. Zumindest legt es die Untersuchung von freien Softwareentwicklungsprojekten nahe, sich auf die Suche nach den Institutionen zu begeben, welche die Rahmenbedingungen für innovatives Handeln setzen und die Entwicklung von Problemlösungsmechanismen zur Bewältigung den Anforderungen der praktischen Entwicklungsarbeit anleiten. In der vorliegenden Untersuchung waren dies die Logik der Gabenökonomie und das Ethos freier Softwareentwicklung, die für die Erklärung der sozialen Konstruktion von freier Software von großer Bedeutung sind.

Teil 3: Theoretischer Ertrag – ein normatives Konzept der Innovationsnetzwerke

An dieser Stelle wollen wir der Frage nach dem theoretischen Ertrag der vorliegenden Untersuchung nachgehen. Im Mittelpunkt steht dabei ein innerhalb der Arbeit immer nur am Rande angesprochenes, aber bislang noch nicht ausführlich diskutiertes Spannungsverhältnis zwischen dem theoretischen Rahmen der Innovationsnetzwerke, der seinem ursprünglichen Konzept nach auf einem Rational-Choice-Handlungskonzept basiert, und unserer bisherigen Argumentation, die starke Anleihen bei normativen Handlungstheorien nimmt. Dies gilt insbesondere für die Auseinandersetzung mit Mauss und Merton bei der Erklärung der Genese des Entwicklungsnetzwerks. Ohne das Ziel auch nur ansatzweise verfolgen zu wollen, die Diskussion zwischen Rational-Choice- und institutionalistischer Handlungstheorien überblicksartig darstellen, wollen wir dieses Resümee dazu nutzen, um die Grundrisse eines institutionalistischen Verständnisses von Innovationsnetzwerken zu entwerfen.

Zu diesem Zweck werden wir erstens die handlungstheoretischen Grundlagen der Transaktionskostenökonomie herausarbeiten, um danach zweitens der Frage nachzugehen, inwieweit das ursprüngliche Konzept der Innovationsnetzwerke diese übernimmt. Hier wird sich ein uneinheitliches Bild ergeben: Wir werden sehen, daß das Konzept implizit auf einer Rational-Choice-Handlungstheorie basiert, allerdings auch an einigen Punkten deutlich darüber hinausgeht. Drittens werden wir ausgehend vom Begriff ‚Institution' und in loser Anknüpfung an ein neo-institutionalistisches Verständnis eine alternative handlungstheoretische

Fundierung entwickeln. Dies bildet für uns dann die Grundlage, um viertens die kategorialen Merkmale von Innovationsnetzwerken Revue passieren zu lassen und der Frage nachzugehen, wie sich das Konzept durch die Wahl eines anderen handlungstheoretischen Ausgangspunkts verändert.

Handlungstheoretische Grundlagen der Transaktionskostenökonomie

Das ursprüngliche Konzept der Innovationsnetzwerke rekurriert – wie weiter oben ausführlicher dargestellt – auf die Transaktionskostenökonomie von Williamson, ein Ansatz, der Strukturalternativen zur Abwicklung von Transaktionen vergleicht. Es will nicht nur die Bedingungen für das Zustandekommen einer bestimmten Form der Abwicklung von Transaktionen angeben, sondern auf Basis der Kenntnis von Rahmenbedingungen, unter denen eine Transaktion abgewickelt wird, die Alternative voraussagen können, die von (rationalen) Akteuren gewählt wird. Im folgenden wollen wir nicht nochmals das Konzept darstellen,[1] sondern die Frage klären, welche handlungstheoretischen Annahmen ihm zugrunde liegen.

Williamson unterscheidet zwischen den institutionellen Arrangements Markt, Hierarchie und intermediäre Formen (Netzwerke), in denen Transaktionen abgewickelt werden können. Diese Alternativen sind jeweils mit verschieden hohen Kosten für Anbahnung, Kontrolle und Vertrag verbunden (Williamson 1996: 180). Das in einer Situation gewählte Arrangement nimmt innerhalb der Theorie die Position der abhängigen, zu erklärenden Variable ein, während die damit verbundenen Kosten die unabhängige Variable darstellen. Die Kosten variieren in Abhängigkeit von (a) den Tauschpartnern zur Verfügung stehenden Informationen, (b) der Unsicherheit der Transaktion, (c) der Wahrscheinlichkeit eines opportunistischen Verhaltens des Transaktionspartners, (d) der Zentralität des getauschten Guts, sowie (e) der Häufigkeit der Transaktion. Durch Kenntnis dieser Faktoren ist es möglich, Voraussagen darüber zu treffen, welche Strukturalternativen gewählt werden, wobei die Transaktionskostenökonomie hier einen dritten Typus von Theoriebestandteil einführt: Dies ist das Selektionskriterium ‚Minimalität von Transaktionskosten', das die Akteure bei Entscheidungen zwischen Strukturalternativen anwenden. Die Akteure in der Transaktionskosten-

1 Vgl. hierzu den theoretischen Rahmen weiter oben.

ökonomie sind also Nutzenmaximierer, die darauf achten, Transaktionen zu möglichst geringen Kosten durchzuführen.

Neben dieser Orientierung am ‚Sparsamkeitsprinzip' trifft das Konzept jedoch noch zwei weitere Aussagen bezüglich der Akteure, die Williamson am deutlichsten im zweiten Kapitel seiner Monographie „The Economic Institutions of Capitalism" (1985) herausstellt: Zum einen ist dies die Annahme, Akteure operieren unter den Bedingungen von ‚bounded rationality', zum anderen jene, dass Akteure zu opportunistischem Verhalten neigen. In bezug auf das erste Merkmal ‚bounded rationality' zitiert er eine berühmte Passage von Herbert Simon: „Bounded rationality is the cognitive assumption on which transaction cost economics relies. This is a semistrong form of rationality in which economic actors are assumed to be ‚intendedly rational, but only limited so' (Simon 1961 p. xxiv). Note the simultaneous reference to both intended and limited rationality." (Williamson 1985: 30) Obwohl Williamson sich hier explizit auf die Annahmen von Simon bezieht, Akteure entschieden unter den Bedingungen von begrenzten Informationen und Zeitknappheit, folgt er ihm nicht darin, die Annahme zu verwerfen, Akteure betrieben bei Entscheidungs- oder Wahlsituationen eine Maximierung von Nutzen. Er lehnt Simons schwächere Unterstellung des ‚satisficing'[2] (Akteure wählen die erstbeste Alternative, die ein bestimmtes Anspruchsniveau befriedigt) ab. Der Verweis auf die Begrenztheit der Akteursrationalität gerät damit zum Gemeinplatz, Akteure seien nicht allwissend und von ihren kognitiven Kapazitäten her nicht in der Lage, sämtliche Informationen gleichermaßen zu berücksichtigen. Durch die Kombination der Annahmen ‚Maximierung von Nutzen' und ‚ungünstige Bedingungen' (bounded rationality) reiht sich Williamson in die Tradition der Rational-Choice-Theoretiker ein, ohne seine Theorie im Fall des Auftretens empirischer Abweichungen von einer Prognose fallen lassen zu müssen: Sollten Akteure einmal eine Strukturalternative zur Abwicklung ihrer Transaktionen wählen, die nicht die kostenminimierende Alternative darstellt, so kann dieser Befund auf die beschränkte Rationalität der Akteure zugerechnet werden.

Unter der zweiten Annahme, Akteure neigen zu opportunistischem Verhalten, versteht Williamson das am eigenen Interesse orientierte Verhalten unter Einschluß von arglistiger Vorteilnahme (Williamson 1985: 47): Die Akteure wählen, sofern sich eine Gelegenheit dazu bietet. Eine arglistig täuschende Handlungsalternative bietet immer diejenige Option, die zumindest kurzfristig den höchsten Gewinn verspricht. Dieser resultiert allerdings nicht aus stark gesenkten Transaktionskosten –

2 Siehe zu diesem Konzept Simon 1957; March 1990.

den Overheads –, sondern aus dem Austausch selbst, also dem Gut oder einem dafür bezahlten Preis. Es stellt sich die Frage, weswegen Williamson ausschließlich an dieser Stelle seinen Rahmen verläßt und sich neben den Kosten der Abwicklung einer Transaktion selbst auch für den Gegenstand der Transaktion interessiert. Die Vermutung liegt hier nahe, daß er mit dieser Annahme die Relevanz von Transaktionskosten unterstreichen möchte: Kontrolle von Transaktionen sind aufgrund der Möglichkeit eines opportunistischen Verhaltens von Transaktionspartnern immer notwendig, weswegen die Beteiligten den Kosten besondere Aufmerksamkeit entgegenbringen.

Einen letzten Aspekt, den wir Zusammenhang mit der Transaktionskostenökonomie thematisieren wollen, bildet die Ebene, auf der sich die Erklärung der Transaktionskostenökonomie bewegt. Als Rational-Choice-Theoretiker verschreibt er sich dem methodischen Individualismus, demnach größere sozialen Strukturen, wie die Strukturalternativen ‚Markt', ‚Netzwerk' und ‚Hierarchie', als erklärungsbedürftige Phänomene wahrgenommen werden, während ausschließlich die Interessen und situativen Faktoren einer Transaktion Erklärungskraft besitzen. Die Erklärung der Transaktionskostenökonomie verläuft also von der Ebene der einzelnen Handlung oder Entscheidung in Richtung der Strukturebene, wobei gegenläufige Wirkungen höchstens in Form der Rahmenbedingungen der Entscheidungen von Akteuren – im Sinne einer Logik (Hill 2002: 24) oder Definition der Situation (Esser 1996: 7) – Berücksichtigung finden. Maßgebliche Richtung der Erklärung ist jedoch ‚bottom-up'.[3]

Grundannahmen der Transaktionskostenökonomie im Konzept der Innovationsnetzwerke?

Welche Konsequenzen hat es nun, daß das Konzept der Innovationsnetzwerke auf die Transaktionskostenökonomie zurückgreift? Übernimmt es die Prämissen von Nutzenmaximierung, begrenzter Rationalität, opportunistischem Verhalten und methodischem Individualismus – und wenn ja, welche Konsequenzen werden durch die Wahl dieses theoretischen Ausgangspunkts eingekauft?[4] Auf den ersten Blick fällt eine

3 Vgl. zum methodischen Individualismus von Rational-Choice-Theorien Coleman 1991: 6 ff.; Peuchert/Scheer 2003: 354; Mozetič 1998: 205.

4 Die Übernahme von handlungstheoretischen Prämissen aus der Rational-Choice-Theorie ist kein Spezifikum des Konzepts der Innovationsnetz-

Absetzbewegung des Konzepts der Innovationsnetzwerke von der Transaktionskostenökonomie ins Auge, die darin besteht, die Fokussierung auf Transaktionskosten zu kritisieren und die Berücksichtigung anderer Formen von Kosten und Nutzen der Kooperation einzufordern. Eine solche Kritik ist innerhalb der Netzwerkforschung weit verbreitet und geht üblicherweise mit der Anmahnung einer Erweiterung des Konzepts einher.[5] Die Vertreter der Theorie der Innovationsnetzwerke interessieren sich ausschließlich für eine bestimmte Form der Abwicklung von Transaktionen – diejenigen, die bei Williamson als ‚intermediäre Form' zwischen Markt und Hierarchie am wenigsten eindeutig beschrieben wird – und fordern eine Erweiterung in Richtung der Berücksichtigung des Nutzens ‚Informationsgewinn mit hoher Innovationsrelevanz' beziehungsweise des Nutzens einer Absorption von Risiken, die ihre Ursprünge in turbulenten Märkten haben. Exemplarisch tritt diese in der folgenden Passage hervor:

„Neben der von Williamson ins Auge gefaßten (effizienten) Transaktionskostenminimierung sind es im Kontext der Innovationsprozesse besonders die technologiestrategischen Vorteile und der Know-how-Transfer (Informationsaustausch, Lernprozesse), die zur Zusammenarbeit von Herstellern und Verwendern führen. […] Technologiestrategische Überlegungen, die in einem gewissen Gegensatz zur Effizienz stehen können, spielen hier eine wichtigere Rolle als die Minimierung von Transaktionskosten. Im Vordergrund der Analyse von Innovationsprozessen steht deshalb nicht die ökonomische Bewertung der Informationsaustauschprozesse, für die Williamson sich interessiert, sondern die Informationsaustauschprozesse sind hier in ihrer Bedeutung für die Reduktion technologischer Unsicherheit und Marktintransparenz relevant." (Kowol 1998: 90)

Mit der Berücksichtigung einer anderen Form von Kooperationsgewinnen werden die handlungstheoretischen Annahmen bezüglich der Akteure nun aber nicht verworfen, sondern eher übernommen: Die Konstitution von Innovationsnetzwerken basiert auf einem bestimmten Typus von Nutzen, nach dessen Realisierung die Akteure streben und wegen dem die Akteure kooperieren. Die Kosten/Nutzen-Rechnung wird durch das Konzept der Innovationsnetzwerke erweitert um die Position ‚innovationsrelevantes Wissen', wobei die Realisierung dieses von den Akteuren

werke, sondern läßt sich bei einer Vielzahl soziologischer Netzwerktheorien beobachten. Vgl. z.B. Weyer 2000b: 241.

5 Vgl. Sydow 1992: 135. Dieser wendet gegen den Transaktionskostenansatz ein, er würde weder Qualitäts- noch Zeitargumente bei der Abwägung zwischen unterschiedlichen Strukturalternativen berücksichtigen.

hoch bewerteten Nutzens dazu führen kann, daß hohe Transaktionskosten in Kauf genommen werden.

Analoge Argumentationsfiguren, die sich ebenfalls im Rahmen einer Theorie rationalen Wahlverhaltens bewegen, finden sich auch in bezug auf ‚Vertrauen'. Innerhalb der langen vorvertraglichen Phase, die dem ‚Weben' und ‚Schließen' der Kooperationsbeziehung dient (Kowol/ Krohn 1997: 49), werden nicht nur die zu erbringenden Leistungen ausgehandelt, sondern auch ein Vertrauensverhältnis aufgebaut. Allerdings hat dieses keinen Eigenwert, sondern besitzt dem Konzept der Innovationsnetzwerke nach instrumentellen Charakter, indem es als Mittel zur Realisierung der gewünschten Handlungsoption dient:

„Vertrauensinvestitionen versprechen für beide Partner einen differentiellen Gewinn, der mit der Strategie des Misstrauens und der Absicherung durch Kaufverträge nicht erreichbar wäre. Die Vertrauensbasierung als Integrationsmodus für Innovationsnetzwerke zielt auf die Etablierung einer rekursiven, prozeduralen Lernstruktur, die Positivsummenspiele für die Partner generiert. Die einmal in Gang gesetzt Dynamik der Vertrauensspirale ist die soziale Basis dafür, dass in rekursiven, selbstverstärkenden Lernprozessen die Technikerzeugung und die Optimierung der Technikanwendung ineinander greifen. […] (A vertraut auf das Verhaltensversprechen von B, das B abgibt, weil er auf das von A vertraut)." (Kowol/Krohn 2000: 139 f.)

Die Bereitschaft der Akteure zu vertrauen und sich damit dem Risiko des Vertrauensbruchs auszusetzen, gründet sich auf die erhofften Kooperationsgewinne, die nur auf der Basis von Vertrauensinvestitionen als Vorleistung beider Akteure zu realisieren sind. Im Konzept der Innovationsnetzwerke wird damit Vertrauen zu einem Kostenfaktor, der – ebenso wie der erstrebte Nutzen von innovationsrelevantem Wissen – nur mit einer bestimmten Wahrscheinlichkeit zu Buche schlägt. Auch diese Kosten ließen sich im Rahmen von Rational-Choice-Ansätzen durch ‚Erwartung mal Wert'-Konstrukte berücksichtigen.[6] Die Auflösung von doppeltkontingenten Situationen[7] und der Aufbau von Vertrauen gelingt nun allerdings nicht auf einen Schlag, sondern durch eine einseitige Vorleistung und Erwiderung dieser, die zu der oben erwähnten

6 In neueren Varianten von Rational-Choice-Theorien wird betont, daß Kalkulationen auf subjektiv *erwartetem*, also unsicherem Nutzen basieren. Siehe hierzu das RREEMM-Modell (resourceful, restricted, evaluating, expecting, maximizing man) bei Esser (1993: 238) und das SEU-Modell (Subjective Expected Utility) bei Hill 2002: 47.

7 Zum Problem doppelter Kontingenz siehe ausführlicher Luhmann 1984: 148 ff., 1997: 332.

Vertrauensspirale führt.[8] Das Problem der Kooperation und der Realisierung von angestrebtem Nutzen wird hier als eine Abfolge von Entscheidungssituationen verstanden, bei der die Kooperationspartner aus den Handlungsalternativen Vertrauen/Mißtrauen wählen. Bei der Erklärung der Kooperation unterstellen die Vertreter des Konzepts den beteiligten Akteuren eine Orientierung an einer tit-for-tat-Strategie (Axelrod 1988), bei der Vertrauen als Vorschuß gewährt und danach Vertrauen mit Vertrauen belohnt und Vertrauensbruch durch Vertrauensentzug bestraft wird.

Allerdings ist unser bisheriger Schluß – das Konzept der Innovationsnetzwerke erbt durch die Wahl der Transaktionskostenökonomie als theoretischen Ausgangspunkt die ihm zugrunde liegenden handlungstheoretischen Annahmen – einseitig, da es an einigen Stellen deutlich über den Rahmen von Rational-Choice-Theorien hinausgeht. Beispiele hierfür bilden die im Konzept beschriebenen länger anhaltenden Kooperationsbeziehungen.

> „Eine weitere wichtige Größe der Vertrauensbasierung sind schließlich die Signale personenzentrierter informeller Kommunikation, über die wechselseitige Verpflichtungen aufgebaut werden (‚facework commitment‘). Bei günstigen Konditionen dieser Art kann sich eine selbstverstärkende Vertrauensbeziehung aufbauen. Wechselseitig wird dann beobachtet, welches Handeln als vertrauenswürdig akzeptiert wird (z.B. Preisgabe von Konstruktionsdaten, strategische Informationen, finanzielle Vorleistungen). Es bilden sich Verhaltensregeln im Netzwerk heraus, deren Beachtung wechselseitig beobachtet wird.“ (Kowol 1998: 322 f.)

Im Verlaufe der Kooperationen werden die Beziehungen in zunehmender Weise multiplex: Sie basieren nicht mehr ausschließlich auf Kooperationsgewinnen, sondern zunehmend auf normativen Erwartungen, wie sich im Zitat an der Betonung von wechselseitigen Verpflichtungen und Verhaltensregeln zeigt. Die Dynamik des Netzwerks läßt sich bei der Fortdauer der Kooperation nicht mehr ausschließlich im Rahmen einer Theorie rationalen Wahlverhaltens erklären, sondern nur durch eine Handlungstheorie, die in der Lage ist, auch diese normativen Komponenten mit zu erfassen.

Auch bei der Beschreibung von lock-in-Effekten verläßt das Konzept die handlungstheoretische Fundierung von Nutzenmaximierung, begrenzter Rationalität und opportunistischen Verhalten im folgenden Zitat, da es sich hierbei um ‚irrationale‘ oder zumindest nicht angestrebte Effekte handelt:

8 Vgl. hierzu auch Kowol/Krohn 1997.

„Wenn die Gestaltung neuer Techniken durch soziale Schließungsprozesse vorangetrieben wird, drohen lock in Effekte. Die Netzwerke setzen ihre Teilnehmer dem Risiko fehlgeleiteter Ressourcen aus (Verriegelung), wenn zu lange Präferenzenwechsel am Markt oder technologisch attraktive Alternativen der Konkurrenten ignoriert werden." (Kowol/Krohn 1997: 59)

Das Risiko eines Festhaltens an gefundenen technischen Lösungen, die mit der Zeit an den Bedürfnissen des Markts vorbeigehen, resultiert aus den sozialen Schließungsprozessen, die eine Kooperation nach sich zieht. Solche Netzwerkexternalitäten länger anhaltender Kooperation, an dessen Auftreten keiner der Akteure ein Interesse haben kann, fügt sich nur schlecht in den Rahmen einer rationalen Handlungstheorie ein. Eine denkbare Erklärung wäre zwar, vom Entstehen einer neuen Präferenz – das Eigeninteresse am Erhalt der Kooperation – auszugehen und anzunehmen, dieses überwiege das Interesse an der Realisierung von Kooperationserträgen, oder zu unterstellen, die switching-costs seien höher als die Kosten der Aufrechterhaltung der Kooperation. Andererseits könnte der Verlust von Wahlfreiheit, der zu einem irrationalen Ergebnis führt, erklärt werden durch eine Veränderung des Fundaments der Kooperation, die nicht mehr auf einem gemeinsamen Interesse an der Realisierung von Kooperationserträgen, sondern auf normativen Erwartungen basiert. Die Netzwerkexternalitäten würden dann daraus resultieren, daß nicht konsequent genug auf der Basis stabiler Präferenzen die Realisierung von Kooperationsgewinnen gewählt wird,[9] sondern an die Stelle der Wahlsituation normenreguliertes Handeln tritt. Wie eine handlungstheoretische Fundierung mit Blick auf diese zweitgenannte Interpretation nun allerdings aussehen könnte, darüber gibt das Konzept der Innovationsnetzwerke keine Auskunft.

Die bisherigen Überlegungen lassen sich in einem Zwischenresümee zusammenfassen: Durch die Wahl der Transaktionskostenökonomie als Ausgangspunkt für die Entwicklung des Konzepts der Innovationsnetzwerke fließen die handlungstheoretischen Annahmen von Nutzenmaximierung, begrenzter Rationalität und opportunistischem Verhalten in das Konzept ein und werden dort weitgehend implizit mitgeführt. Dies gilt insbesondere für die Konstitution des Netzwerks, die durch komplementäre Interessen und rationale Wahl erklärt wird. Gegenüber der Transaktionskostenökonomie ist das Konzept jedoch insofern anspruchsvoller, da es andere Formen von Erträgen als die Minimierung von Transaktionskosten mit berücksichtigt: strategische Informationsgewinne und innovationsrelevantes Wissen, deren Erzielung und Wert im vornherein

9 Gemeint ist hier die Wahl zwischen den Alternativen ‚Fortführung der Kooperation' und ‚Abbruch der Kooperation'.

schwer zu kalkulieren und deren Realisierung mit erheblichen Kooperationsrisiken verbunden ist. Mit dieser Fokussierung zielt das Konzept auf die Schließung der Lücke, welche die Transaktionskostenökonomie bei der Erklärung von Innovationsprozessen gelassen hat.

Weiter basiert das Konzept der Innovationsnetzwerke zwar auf den handlungstheoretischen Annahmen des rationalen Akteurs, reichert diese aber durch andere Komponenten an, deren handlungstheoretische Bezugspunkte allerdings im Dunkeln liegen. So gilt doch auch hier, wie häufig in der allgemeinen Literatur zu sozialen Netzwerken, daß es an einem „soziologisch durchgearbeiteten Akteurskonzept [fehlt, N.C.T.], welches das Verhältnis von strategischem Eigeninteresse und Kooperationsorientierung analytisch in den Griff bekommt" (Weyer 2000b: 241).

Theoretische Konkurrenz: institutionalistische Handlungstheorie

An dieser Stelle wollen wir knapp Rückschau halten und der Frage nachgehen, wie sich diese handlungstheoretischen Annahmen mit den empirischen Ergebnissen der vorliegenden Untersuchung vertragen. Im zweiten Teil der Arbeit haben wir gesehen, daß das Netzwerk freier Softwareentwicklung ihre Existenz zwei Innovationen verdankt: (a) die vertragliche Innovation freier Softwarelizenzen und (b) die organisatorische Innovation der offenen Programmierprojekte. Mit der ersten Innovation verbunden ist eine Perspektive auf den Zusammenhang von Werk, Werkschöpfer und Werknutzer, eine vertragliche Regulation, die freie Software zum Kollektivgut macht, sowie normative Erwartungen in bezug auf Werkschöpfer, die eine Gabenökonomie konstituieren. Die zweite Innovation beinhaltet eine neuartige Perspektive auf die Organisation der Entwicklung von Software sowie ein Ethos, das sich aus vier Normen zusammensetzt. Bei der anschließenden empirischen Analyse freier Softwareprojekte wurde dann deutlich, daß die Entwickler zum großen Teil normativ motiviert sind, Diskussionen über Zielsetzungen im Modus der Argumentation geführt werden, bei der eine Begründungspflicht existiert, und die Entwickler-Anwender-Rückkopplungen, die zu einer hohen Stabilität von freier Software führen, auf Rollen mit klaren normativen Erwartungen basieren. Diese Befunde legen es nahe, für den hier interessierenden Fall von einer anderen, normativ-institutionalistischen handlungstheoretischen Grundlage auszugehen und von dort aus eine alternative Lesart des Konzepts der Innovationsnetzwerke zu entwickeln.

Einen Ausgangspunkt für die Entwicklung einer den Ergebnissen der Untersuchung angemesseneren handlungstheoretischen Fundierung des Konzepts der Innovationsnetzwerke gewinnen wir mit dem Begriff ‚Institution'. Aufgrund der Vielfalt der Bedeutungen des Begriffs bedarf es hier einiger Klärungen: Eine erste Abgrenzung bezieht sich auf ältere institutionalistische Konzepte, in denen Institutionen als normative Orientierungen konzipiert werden, die ausschließlich handlungsbeschränkenden Charakter besitzen. Hier nehmen Institutionen den Charakter von Leitplanken des sozialen Handelns an, die den Alternativenspielraum und die Wahlfreiheit von Akteuren beschneiden. Diese Bestimmung von Institution ist zwar in vielen Fällen zutreffend, bleibt aber insofern einseitig, als daß der Aspekt der Ermöglichung von Handlungen unterschlagen wird. In neueren Konzepten werden daher neben der Beschneidung des Spielraums mindestens ebenso stark deren handlungsermöglichende Wirkungen betont (Scott 1995: 38; Krücken 2002: 5).[10]

Zweitens verstehen wir unter Institutionen nicht ausschließlich normative Erwartungen. In Anlehnung an neo-institutionalistische Konzepte gehen wir davon aus, daß Institutionen das Akteurshandeln in umfassender Weise orientieren, indem sie neben normativen auch kognitive[11] und regulative Komponenten beinhalten (Meyer/Rowan 1991: 42). Während die von Institutionen bereitgestellten kognitiven Komponenten für eine Vereinheitlichung der Perspektive von Akteuren auf einen Gegenstand sorgen, leiten regulative Verhalten an, indem sie Regeln setzen, überwachen und sanktionieren (Scott 1995: 35).

Zu klären ist drittens das Verhältnis zwischen Institutionen und individuellem Handeln bzw. der Grad an Einfluß, den Institutionen auf Handeln ausüben. Notwendig ist dies, da innerhalb einer institutionalistisch orientierten Soziologie bisweilen die These der handlungsdeterminierenden Wirkungen von Institutionen entwickelt wird,[12] die wir nicht teilen. Im Unterschied zu diesen ‚harten' normativen Konzepten gehen wir davon aus, daß Institutionen Handlung und Entscheidung rahmen, jedoch im Regelfall den Akteuren Spielräume und Wahlfreiheiten lassen. Demnach sind sie dafür verantwortlich, in welcher Weise sich Entscheidungssituationen für den Akteur darstellen, welche Optionen überhaupt als Entscheidungsalternativen wahrgenommen werden, was Nutzen und was Kosten sind beziehungsweise welche Effekte als erwünscht und welche als nicht erwünscht gelten. Institutionalisierte Vorgaben müssen also von den Akteuren aktiv verarbeitet werden (DiMaggio/Powell

10 Vgl. auch Giddens 1988: 89.

11 Zu einer mikrotheoretischen Fundierung eines kognitiv angereicherten Verständnisses von ‚Institution' siehe Zucker 1991: 103 ff.

12 Oder es zumindest diesen Konzepten unterstellt wird.

1991: 22) und stellen im Regelfall keine Orientierung dar, die genügend Eindeutigkeit besitzt, um das Handeln und Entscheiden in einer konkreten Situation zu orientieren. So betont beispielsweise Scott (1995: 39), daß die einer Situation angemessene oder ‚passende' Orientierung häufig gewählt werden muß. Zudem sind unterschiedliche institutionelle Vorgaben nicht immer konsistent, sondern befinden sich häufig in einem Spannungsverhältnis oder sind gar widersprüchlich. Solche Konstellationen ermöglichen es den Akteuren, die Widersprüchlichkeit instrumentell zu nutzen (Hasse/Krücken 1999: 59).

Weiter ist zu beobachten, daß Verhaltensvorgaben selten auf eine Situation einfach angewendet werden können. Vielmehr ergeben sich häufig Überraschungen und Uneindeutigkeiten, die zu Handlungsspielräumen für die beteiligten Akteure führen. Wir halten also fest: Institutionen steigern in einer Vielzahl von Fällen die Wahrscheinlichkeit für ein bestimmtes Verhalten. Als wie stark sich allerdings der Einfluß der Institutionen auf das Handeln der Akteure erweist, ist letztlich eine empirische Frage. Hierfür bestimmend sind die Eindeutigkeit normativer Erwartungen und deren Anwendung auf Handlungssituationen, das etwaige Vorhandensein von konkurrierenden Institutionen sowie die Stärke der Verankerung einer Institution im betreffenden Feld.

Innovationsnetzwerke revisited: Ausblicke auf eine institutionalistische Lesart

Nach diesen Klärungen zum Begriff Institution haben wir einen anderen handlungstheoretischen Ausgangspunkt gewonnen, von dem aus wir einige zentrale Bestandteile des Konzepts der Innovationsnetzwerke durchgehen und eine alternative Lesart entwickeln können. Dies sind im einzelnen die Fragen (a) nach der Konstitution von Innovationsnetzwerken, (b) nach dem Zugang zu Netzwerkressourcen, (c) nach dem Stellenwert von Vertrauen sowie (d) nach der Bedeutung des Kooperationsrisikos lock-in-Effekt.

Eine wesentliche Differenz zwischen dem ursprünglichen Konzept der Innovationsnetzwerke und der hier entwickelten institutionalistischen Lesart bezieht sich auf die Konstitution des Netzwerks. Wie weiter oben ausführlicher gezeigt, geht das ursprüngliche Konzept im jeweiligen Fall von Kooperationserträgen aus, an deren Realisierung die Akteure interessiert sind und die diese zur Kooperation motivieren. Unsere empirischen Ergebnisse legen es dagegen nahe, Netzwerkkonstitution mit Verweis auf Institutionen zu erklären, wobei in diesem Prozeß Solidaritäts-, Reziprozitäts- und Kooperationsnormen denjenigen normati-

ven Orientierungshorizont darstellen, der die Konstitution der Kooperationen begünstigt. Wie hat man sich diesen Einfluß nun vorzustellen? Instruktiv ist hier der Fall freier Softwareentwicklung: Wie gezeigt beinhaltete der für die Entwicklung von freier Software maßgebliche Orientierungshorizont ein Ethos, ohne den die Kooperation schwer vorstellbar ist. Die Parallele zwischen dem Ethos freier Softwareentwickler und dem wissenschaftlichen Ethos ist nun alles andere als zufällig. Sie ist dadurch zu erklären, daß die Gruppe der Entwickler von freier Software ihren Ursprung in den Rechenzentren amerikanischer Universitäten hat und es plausiblerweise angenommen werden kann, daß sie dort mit dem wissenschaftlichen Ethos vielfach in Kontakt gekommen ist.[13] Das gilt auch heute noch für eine Vielzahl von freien Softwareentwicklern, die Studenten sind und damit bereits eine akademische Sozialisation genossen haben.

Für ein institutionalistisches Konzept von Innovationsnetzwerken läßt sich aus diesen Befunden die folgende Hypothese ableiten: Eine Konstitution von Netzwerken basiert auf dem Transfer von normativen, kognitiven und regulativen Komponenten, die aus der Umwelt der Akteure stammen und an denen sich die kooperierenden Akteure orientieren. Von zentraler Bedeutung sind Reziprozitäts- und Solidaritätsnormen sowie normative Gemeinwohlorientierungen. Sobald diese für die Akteure maßgeblich werden, sich in der Kooperation bewähren und Verhaltenserwartungen an andere nicht dauerhaft enttäuscht werden, kann es zu einer Stabilisierung dieser Beziehungen kommen. Unsere erste These lautet also:

Die Konstitution normativ basierter, also auf Kooperations- und Solidaritätsnormen gründender Netzwerke ist überall dort wahrscheinlich, wo in der Umwelt der Akteure solche institutionalisierten Orientierungen vorzufinden sind und als Ressource für den Aufbau von Kooperationsbeziehungen fruchtbar gemacht werden.14

---

13 Vgl. zu den Ursprüngen der Unix-Entwicklung im akademischen Bereich McKusick 1999; zur Entstehung des GNU Projekts am MIT Stallman 1999, 2002: 15 ff.

14 Dieses Konzept eines Strukturaufbaus vermittels institutionalisierter Orientierungen aus der Umwelt sollte nicht verwechselt werden mit dem Konzept der mimetischen Isomorphie, wie es im Neo-Institutionalismus vorzufinden ist (DiMaggio/Powell 1983). Grundlegende Annahme ist dort, daß soziale Systeme – wie z.B. Organisationen – auf institutionalisierte Mythen innerhalb ihrer Umwelt reagieren und dazu neigen, die Grenze zwischen System und Umwelt ‚verschwinden' zu lassen, indem sie legitimierende Elemente in die eigene legitimatorische Struktur aufnehmen (Meyer/Rowan 1991: 47). Im Unterschied zu neo-institutionalistischen Interpretationen gehen wir nun allerdings nicht zwangsläufig von einem hohen Grad an Unabhängigkeit zwischen legitimatorischer und funktionaler

Richtet man seinen Blick auf die Frage nach der Regulation des Zugangs zu den innerhalb des Innovationsnetzwerks produzierten Ressourcen, so stellt man fest, daß die Netzwerke des Werkzeugmaschinenbaus, an dem das ursprüngliche Konzept der Innovationsnetzwerke entwickelt wurde, sich an einem Extrem befindet, da es den Zugriff auf die Ressource ‚innovationsrelevantes Wissen' stark restriktiv handhabt, während sich der Fall freier Softwareentwicklungsprojekte am anderen Extrem befindet, da dort jedermann über den Zugang zum Gut verfügt. Handelt es sich im einen Fall um ein exklusives Gut, bildet es im anderen Fall ein Kollektivgut, das jedermann zugänglich ist.[15] Welche Faktoren sind nun dafür verantwortlich, wie der Zugang zu den im Netzwerk produzierten Ressourcen reguliert wird? Wir gehen im Hinblick auf diese Eigenschaft des Netzwerks davon aus, daß die Form der Zugangsregulierung nur durch die im Feld relevanten Institutionen erklärt werden kann: Hierzu zählen im Fall von freier Software als regulative Komponenten die Lizenzbestimmungen, mit denen der Zugang zur Netzwerkressource für sämtliche Nutzer auf Dauer gestellt wird. Von ebenso großer Bedeutung ist aber die oben analysierte kognitive Dimension der Institution, die Perspektive auf Werk, Werkschöpfung und Nutzung, sowie die von dieser Perspektive abgeleiteten normativen Komponenten. Diese institutionalisierten Orientierungen sind dafür verantwortlich, daß die Netzwerkressource als Kollektivgut betrachtet und durch die regulative Komponente die freie Zugänglichkeit herbeigeführt wird. Bezüglich des Zugangs zur Netzwerkressource wollen wir die folgende Hypothese formulieren: Je stärker ein produziertes Gut der institutionalisierten Orientierung nach als individuelle, strategische Ressource gilt, desto restriktiver wird der Zugang reguliert, je stärker ein Gut als Kollektivgut oder im kollektiven Interesse liegend interpretiert wird, desto freizügiger werden die Nutzungsmöglichkeiten hier ausfallen.

Eine weitere wesentliche Differenz zwischen dem ursprünglichen Konzept der Innovationsnetzwerke und dem hier untersuchten Fall bezieht sich auf den Stellenwert von Vertrauen. Im ursprünglichen Konzept bildet riskanter Vertrauensvorschuß den Einstieg in eine ‚Vertrauensspirale', die eine wesentliche Rolle bei der Netzwerkkonstitution spielt. Der hohe Stellenwert von Vertrauen findet sich auch in der allgemeineren Literatur zu sozialen Netzwerken wieder, wo es ebenso wie im Konzept der Innovationsnetzwerke als Koordinationsmittel diskutiert

Struktur von sozialen Systemen aus. Oben meinen wir den Einbau von Elementen aus der Umwelt in die funktionale Struktur.

15 Mit Blick auf andere Netzwerke wird deutlich, daß zwischen vollständig restriktivem und vollständig freizügigem Zugang zur Netzwerkressource eine Vielfalt von Formen existiert.

wird (z.B. Weyer 2000a: 7; Willke 1995: 146). Im starken Kontrast hierzu spielt in unserer Untersuchung Vertrauen keine herausragende Rolle: Weder finden sich in den durchgeführten Interviews Hinweise auf einen solchen zentralen Stellenwert,[16] noch ist aus der Kommunikation auf der Projekt-Mailingliste ein solcher abzuleiten. Worin liegt nun die Ursache für diese Diskrepanz bzw. wie läßt sich dieser Befund erklären? Vertrauen kommt immer dann ins Spiel, wenn bestimmte Handlungsweisen von alter als unerwünscht gelten und für ego die Handlungsalternativen ‚Vertrauen' und ‚Mißtrauen' verfügbar sind.[17] Im ursprünglichen Konzept wird Vertrauen hinsichtlich der Behandlung von sensiblen Informationen und innovationsrelevantem Wissen virulent. Sucht man nach ähnlichen Problemstellungen für die Akteure im Bereich der freien Software, wird man enttäuscht: Freie Softwarelizenzen erlauben einen unbeschränkten Zugriff auf die Ressourcen, gestatten Vervielfältigung, Nutzung, Modifikation und Vertrieb. Hier gilt also ein Großteil von Handlungen – wie z.B. die Nutzung von Netzwerkressourcen, ohne zur Produktion beigetragen zu haben – nicht als unerwünscht, so daß nicht auf eine Unterlassung vertraut werden muß. Die andernorts als opportunistisch geltenden Verhaltensweisen sind hier legitim.

Eine Ausnahme bildet – zumindest im Fall von ‚copyleft-Lizenzen'– die Aneignung von freier Software. Hierunter zu verstehen ist der Vertrieb einer modifizierten oder nicht modifizierten Version des Programms unter anderen als den ursprünglichen Lizenzbedingungen. Eine solche Privatisierung, auch ‚hijacking' genannt (O'Mahony 2003: 1181), bildet nun aber eine unerwünschte Handlung, auf deren Unterlassung gerade nicht vertraut werden kann: Da freie Software unbeschränkt zugänglich ist und eine Aneignung von jedermann betrieben werden kann, müßte nun auch jedermann vertraut werden. An die Stelle eines solchen *unmöglichen Vertrauens* tritt die rechtliche Regulation. Das Verbot der

16 Gerade weil sich bei der Analyse der Projekt-Mailingliste KMail keine Hinweise darauf ergaben, daß Vertrauen eine wichtige Rolle für die Kooperation der Entwickler spielt, wurden die Interviewpartner hierzu explizit befragt. Auch aus den Antworten ließen sich keine Hinweise darauf entnehmen, daß Vertrauen innerhalb von freien Programmierprojekten einen hohen Stellenwert besitzt. Dies gilt sowohl für die im ursprünglichen Konzept der Innovationsnetzwerke thematisierte Rolle von Vertrauen hinsichtlich der Behandlung von sensiblen Informationen (Vertrauensschutz) als auch in bezug auf die Kompetenz der Kooperationspartner (Kowol/Krohn 1997: 57). Während Kompetenz aufgrund der Transparenz des Quellcodes von kompetenten Entwicklern beurteilt und Qualität durch Peer-Review gesichert wird, ist ein Schutz von sensiblen Informationen nicht notwendig, da diese nahezu ausnahmslos öffentlich sind.

17 Wenn keine Handlung unerwünscht ist, ist das Vertrauen auf ihre Unterlassung irrelevant.

Handlung ‚Aneignung von freier Software‘ in freien Softwarelizenzen sorgt dafür, daß die als unerwünscht geltenden Handlungen unterbleiben oder das Auftreten durch das Rechtssystem sanktioniert wird.[18]

Auch diese Überlegungen zum Stellenwert von Vertrauen verweisen auf einen hohen Einfluß der relevanten Institutionen: Der institutionalisierte Orientierungshorizont ist, wie der Fall freier Software zeigt, maßgeblich dafür verantwortlich, welche Rolle Vertrauen innerhalb von Netzwerken spielt. Eine wichtige Rolle von Vertrauen ist überall dort zu vermuten, wo eine Vielzahl von Handlungen als unerwünscht gilt, eine Unterlassung jedoch aus praktischen Gründen nicht durch härtere Mechanismen (wie beispielsweise durch Verträge) garantiert werden kann. Fällt hingegen der Bereich von unerwünschten Handlungen klein aus – beispielsweise indem die Nutzung von Netzwerkressourcen im Sinne eines Kollektivguts jedermann gestattet ist –, verliert Vertrauen an Bedeutung. Der oben analysierte institutionalisierte Orientierungshorizont von Gabenökonomie und Ethos erweist sich auch in bezug auf die Erklärung des (geringen) Stellenwerts von Vertrauen als von großer Bedeutung.

Einen letzten Punkt, auf den wir im Zuge unserer Durchsicht der Merkmale von Innovationsnetzwerken eingehen wollen, bezieht sich auf das Risiko von ‚lock-in‘-Effekten. Wie oben gezeigt, tritt der Effekt des Festhaltens an gefundenen Lösungen und das ‚Übersehen‘ von Marktsignalen im Fall von längerfristigen Kooperationsbeziehungen durch soziale Abschließungsprozesse auf. Das Risiko, ‚Ladenhüter‘ zu produzieren, resultiert also aus der Exklusivität des Netzwerks, die dem ursprünglichen Konzept nach garantiert sein muß, damit die Aussicht auf in Netzwerken produzierte Kooperationsgewinne, die Vorteile gegenüber Konkurrenten darstellen, als Anreiz für die Kooperation fungieren kann. Im Unterschied dazu sind Softwareentwicklungsprojekte prinzipiell jedermann zugänglich. Diese Eigenschaft läßt vermuten, daß lock-in-Effekte kein Risiko darstellen, da neu hinzukommende Entwickler neue Ideen und Zielsetzungen in die Projekte tragen. Auch diese Netzwerkeigenschaft erklärt sich aus den für das Feld relevanten Orientierungen: Die Produktion von nicht-exklusiven Ressourcen innerhalb des Netzwerks bildet die Voraussetzung für eine nicht restriktive Handhabung des Zugangs zu ihm. Dies bildet dann wiederum den Grund dafür, daß lock-in-Effekte nicht auftreten. Das Ausbleiben von Verriegelungseffekten folgt also hier letztlich aus der institutionalisierten Ökonomie der Gabe.

18 Verstöße gegen freie Softwarelizenzen stellen eine regelmäßige Begleiterscheinung dar. Vgl. hierzu und zu den Mitteln, die freie Softwareentwickler zur Sanktionierung ergreifen, ausführlicher O’Mahony 2003: 1188.

Literatur

Aaron, J.D., 1970: Estimating resources for large programming systems. In: J.N. Buxton/B. Randell, B. (Hrsg.): Software Engineering Techniques. Report on a conference sponsored by the NATO Science Committee. Rome 27th to 31st October 1969.

Abels, H., 2001: Einführung in die Soziologie. Opladen: Westdeutscher Verlag.

Arnold, V., 1992: Theorie der Kollektivgüter. München: Franz Vahlen.

Asdonk, J./Bredeweg, U./Kowol, U., 1991: Innovation als rekursiver Prozeß. Zur Theorie und Empirie der Technikgenese am Beispiel der Produktionstechnik. In: Zeitschrift für Soziologie 20/4.

Asdonk, J./Bredeweg, U./Kowol, U., 1993: Innovation, Organisation und Facharbeit. Rahmenbedingungen und Perspektiven betrieblicher Technikentwicklung. Bielefeld: Kleine. Wissenschafts- und Technikforschung, Report 40.

Asdonk, J./Bredeweg, U./Kowol, U., 1994: Evolution in technikerzeugenden und technikverwendenden Sozialsystemen – dargestellt am Beispiel des Werkzeugmaschinenbaus. In: W. Rammert/G. Bechmann (Hrsg.): Technik und Gesellschaft, Jahrbuch 7. Frankfurt; New York: Campus.

Axelrod, R., 1988: Die Evolution der Kooperation. München: Oldenbourg.

Baker, F.T., 1979: Chief Programmer Team Management of Production Programming. In: E.N. Yourdon (Hrsg.): Classics in Software Engineering. New York: Yourdon.

Baker, T./Mills, H.D., 1973: Chief Programmer Teams. In: Datamation 19/12.

Balzert, H., 1982: Die Entwicklung von Software-Systemen: Prinzipien, Methoden, Sprachen, Werkzeuge. Mannheim: Bibliographisches Institut.

Barnes, S.B./Dolby, G.A., 1972: Das wissenschaftliche Ethos. Ein abweichender Standpunkt. In: P. Weingart (Hrsg.): Wissenschaftssoziologie I. Frankfurt: Fischer.

Baym, N., 1995b: The Emergence of Community in Computer-Mediated Communication. In: S.G. Jones (Hrsg.): Cybersociety. Computer-Mediated Communication and Community. Thousand Oaks u.a.O.: Sage.

Bechmann, G. (Hrsg.), 1996: Praxisfelder der Technikfolgenforschung. Frankfurt; New York: Campus.

Bechmann, G. 1997: Diskursivität und Technikgestaltung. In: S. Köberle/F. Gloeden/L. Hennen (Hrsg.): Diskursive Verständigung. Mediation und Partizipation in Technikkontroversen. Baden-Baden: Nomos.

Beck, U., 1988: Gegengifte. Die Organisierte Unverantwortlichkeit. Frankfurt: Suhrkamp.

Beck, U./Bonß, W./Lau, C., 2001: Theorie reflexiver Modernisierung – Fragestellungen, Hypothesen, Forschungsprogramme. In: U. Beck/W. Bonß (Hrsg.): Die Modernisierung der Modernisierung. Frankfurt: Suhrkamp.

Beck, U./Giddens, A./Lash, S., 1996: Reflexive Modernisierung. Eine Kontroverse. Frankfurt: Suhrkamp.

Berking, H., 1996: Schenken. Zur Anthropologie des Gebens. Frankfurt; New York: Campus.

Bien, G., 1968: Das Theorie-Praxis-Problem und die politische Philosophie bei Platon und Aristoteles. In: M. Müller (Hrsg.): Philosophisches Jahrbuch 76. Freiburg; München: Karl Alber.

Bijker, W.E., 1987: The Social Construction of Bakelite: Towards a Theory of Invention. In: W.E. Bijker/T. Hughes/T. Pinch (Hrsg.): The Social Construction of Technological Systems – New Directions in Sociology and History of Technology. Cambridge u.a.O.: MIT-Press.

Bijker, W.E., 1995: Of Bicycles, Bakelites and Bulbs. Toward a Theory of Sociotechnical Change. Cambridge u.a.O.: MIT Press.

Bijker, W.E./Law, J., 1992: General Introduction. In: W.E. Bijker/J. Law (Hrsg.): Shaping Technology/Building Society. Cambridge u.a.O.: MIT Press.

Bijker, W.E./Pinch, T., 1984: The Social Construction of Facts and Artifacts: or How the Sociology of Science and the Sociology of Technology might Benefit each other. In. Social Studies of Science 14.

Bijker, W., Pinch, T., 1987: The Social Construction of Facts and Artifacts: Or How the Sociology of Science and the Sociology of Technology Might Benefit Each Other. In: dies (eds.), The Social Construction of Technological Systems. New Direction in the Sociology of Technology. Cambridge, MA: MIT Press.

Böhm, B.W., 1979: Software-Engineering. In: E.N. Yourdon (Hrsg.): Classics in Software Engineering. New York: Yourdon.

Böhm, B.W., 1986: A Spiral Model of Software Development and Enhancement. In: ACM Sigsoft Software Engineering Notes 11/4.

Bonaccorsi, A./Rossi, C., 2002: Licensing schemes in the production and distribution of Open Source software. An empirical investigation. Institute for Informatics and Telematic: Working Paper. Als Online Dokument: http://opensource.mit.edu/papers/bonaccorsirossi license.pdf.

Bonaccorsi, A./Rossi, C., 2003: Why Open Source software can succeed. In: Research Policy 32.

Bora, A., 1999: Differenzierung und Inklusion. Partizipative Öffentlichkeit im Rechtssystem moderner Gesellschaften. Baden-Baden: Nomos.

Bramann, K.-W./Merzbach, J./Münch, R., 1995: Sortiments- und Verlagskunde. 2. überarb. Aufl. München u.a.O.: K.G. Sauer.

Braun, A., 1998: Typisierung von Handlungsformen in der Informatik. In: D. Siefkes/P. Eulenhofer/H. Stach/K. Städtler (Hrsg.): Sozialgeschichte der Informatik. Kulturelle Praktiken und Orientierungen. Wiesbaden: Deutscher Universitätsverlag.

Bredeweg, U./Kowol, U./Krohn, W., 1994: Innovationstheorien zwischen Technik und Markt. In: W. Rammert/G. Bechmann (Hrsg.): Technik und Gesellschaft. Jahrbuch 7. Frankfurt u.a.O.: Campus.

Brodbeck, M., 1994: Software-Entwicklung: Ein Tätigkeitsspektrum mit vielfältigen Kommunikations- und Lernanforderungen. In: C. Frese/M. Brodbeck (Hrsg.): Produktivität und Qualität in Software-Projekten. Psychologische Analyse und Optimierung von Arbeitsprozessen in der Software-Entwicklung. München; Wien: Oldenbourg.

Brodbeck, M., Frese,C., (Hrsg.) 1994: Produktivität und Qualität in Software-Projekten. Psychologische Analyse und Optimierung von Arbeitsprozessen in der Software-Entwicklung. München; Wien: Oldenbourg.

Brooks, F.P., 1975: The Mythical Man Month: Essays on Software Engineering. Reading u.a.O.: Addison-Wesley.

Brooks, F.P., 1995: The Mythical Man Month: Essays on Software Engineering. Anniversary Edition. Reading u.a.O.: Addison-Wesley.

Browne, C.B., 1998: Linus and Decentralized Development. In: First Monday 3/3. Als Online-Dokument: http://firstmonday.org/issues/issue3_3/browne/index.html.

Busch, C., 1998: Zur Bedeutung von Metaphern in der Entwicklung der Informatik. In: D. Siefkes/P. Eulenhöfer/H. Stach/H. Städtler (Hrsg.): Sozialgeschichte der Informatik. Kulturelle Praktiken und Orientierungen. Wiesbaden Deutscher Universitäts-Verlag.

Butler, B./Sproull, L./Kiesler, S./Kraut, R., 2002: Community Efforts in Online-Groups: Who does the Work and Why? Als Online-Dokument: http://opensource.mit.edu/papers/butler.pdf.

Campbell-Kelly, M., 1995: Development and Structure of the International Software Industry, 1950-1990. Als Online-Dokument: http://www.dcs.warwick.ac.uk/~mck/Personal/SoftIndy.pdf.

Campbell-Kelly, M./Aspray, W., 1996: Computer. A History of the Information Machine. New York: Basic Books.

Ceruzzi, P.E., 2000: A History of Modern Computing. Cambridge u.a.O.: MIT Press.

Chroust, G., 1992: Modelle der Software-Entwicklung. München; Wien: Oldenbourg.

Ciborra, C.U./Andreu, R., 2001: Sharing Knowledge across Boundaries. In: Journal of Information Technology 16/2.

Clausen, G., 1991: Schenken und Unterstützen in Primärbeziehungen. Materialien zu einer Soziologie des Schenkens. Frankfurt u.a.O.: Lang.

Coase, R.H., 1937: The Nature of the Firm. Economica, New Series 4/16.

Cole, S., 1972: Wissenschaftliches Ansehen und die Anerkennung wissenschaftlicher Leistungen. In: P. Weingart (Hrsg.): Wissenschaftssoziologie I. Frankfurt: Fischer.

Coleman, J.S., 1991: Grundlagen der Sozialtheorie. Handlungen und Handlungssysteme. München: Olenbourg.

Collingridge, D., 1981: The Social Control of Technology. Stratford: The Open University Press.

Cook, T.D./Campbell, D.T., 1979: Quasi-Experimentation. Design & Analysis Issues for Field Settings. Chicago: Mc. Nally.

Crozier, M./Friedberg, E., 1979: Macht und Organisation: Die Zwänge kollektiven Handelns. Königstein: Athenäum.

Dalle, J.M./Jullien, N., 2002: ‚Libre‘ Software: Turning fads into Institutions?‘ Als Online-Dokument: http://opensource.mit.edu/papers/Libre-Software.pdf.

Denert, E., 1992: Software-Engineering: methodische Projektabwicklung. 1. korr. Nachdr. Berlin u.a.O.: Springer.

DiBona, C./Ockman, S./Stone, M. (Hrsg.), 1999: Open Sources. Voices from the Open Source Revolution. Bejing u.a.O.: O'Reilly.

Dierkes, M., 1993: Möglichkeiten der Technikfolgenabschätzung. In: M. Dierkes (Hrsg.): Die Technisierung und ihre Folgen – Zur Biographie eines Forschungsfeldes. Berlin: Sigma.

Dierkes, M./Hoffmann, U./Marz, L., 1992: Leitbild und Technik. Zur Entstehung und Steuerung technischer Innovation. Berlin: Sigma.

Dierkes, M./Hoffmann, U./Marz, L., 1998: Technikgenese und organisatorischer Wandel. Divergierende Innovationsschemata. In: OECD (Hrsg.): Technologien des 21. Jahrhunderts. Herausforderungen einer dynamischen Zukunft. Paris: OECD.

Dierkes, M./Marz, L., 1994: Leitbildprägung und Leitbildgestaltung. In: G. Bechmann/T. Petermann (Hrsg.): Interdisziplinäre Technikforschung. Genese, Folgen, Diskurs. Frankfurt u.a.O.: Campus.

Dijkstra, E., 1970: Structured Programming. In: J.N. Buxton/B. Randell (Hrsg.): Software Engineering Techniques. Report on a conference sponsored by the NATO Science Committee Rom, 27th to 31st October 1969.

Dijkstra, E., 1979a: Programming Considered as a Human Activity. In: E.N. Yourdon (Hrsg.): Classics in Software Engineering. New York: Yourdon.

Dijkstra, E., 1979b: The Humble Programmer. In: E.N. Yourdon (Hrsg.): Classics in Software Engineering. New York: Yourdon.

DiMaggio, P.J./Powell, W.W., 1983: The Iron Cage Revisited. Institutional Isomorphism and Collective Rationality in Organizational Fields. In: American Sociological Review 48.

DiMaggio, P.J./Powell, W.W., 1991: Introduction. In: P.J. DiMaggio/W.W. Powell (Hrsg.): The New Institutionalism in Organizational Analysis. Chicago; London: The University of Chicago Press.

Döhl, V./Kratzer, N./Moldaschl, M./Sauer, D., 2001: Auflösung des Unternehmens? In: U. Beck /W. Bonß (Hrsg.): Die Modernisierung der Modernisierung. Frankfurt: Suhrkamp.

Donaldson, J.R., 1973: Structured Programming. In: Datamation 19/12.

Douglas, M./Wildavsky, A., 1982: Risk and Culture. Berkeley u.a.O.: University of California Press.

Dumke, R., 1993: Modernes Software Engineering. Eine Einführung. Braunschweig; Wiesbaden: Vieweg.

Dyers, R. 1999: Making white people white. MacKenzie, D.A./Wajcman, J. (Hrsg.) 1999: The Social Shaping of Technology. Women's Technobiographies. 2. Aufl. Buckingham: Open University Press.

Elster, J., 1986: Rational Choice. Oxford: Basic Blackwell Ltd.

Endres, A., 1996: A Synopsis of Software Engineering History: The Industrial Perspective. In: A. Brennecke/R. Keil-Slawik (Hrsg.): History of Software Engineering. Position Papers for Dagstuhl Seminar 9635. August 26th-30th 1996.

Endres, A. 2001: Wem nutzen und wem schaden Software-Patente? In: Informatik Spektrum 2001, Vol. 24, Nr. 1.

Endres, A./Rombacher, D., 2003: A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories. Addison-Wesley: Pearson.

Esser, H., 1993: Soziologie. Frankfurt; New York: Campus.

Esser, H., 1996: Die Definition der Situation. In: Kölner Zeitschrift für Soziologie und Sozialpsychologie 48/1.

Fischer, H.-J., 1993: Urheberrechtsvertrag. München: Beck.

Franke, N./Hippel, E.v., 2003: Satisfying heterogenous user needs via innovation toolkits: the case of Apache security software. In: Research Policy 32.

Free Software Foundation (FSF), o.J.: The Free Software Definition. Als Online-Dokument: http://www.gnu.org/philosophy/free-sw.html.

Fromm, F.K./Nordemann, W., 1998: Urheberrecht. Kommentar zum Urheberrechtsgesetz und zum Urheberrechtswahrnehmungsgesetz. Stuttgart u.a.O.: Kohlhammer.

Garton, L./Haythronthwaite, C./Wellman, B., 1997: Studying Online Social Networks. In Journal of Computer-Mediated Communication 3/1.

Garzarelli, G., 2002: Open Source Software and the Economics of Organization. Als Online-Dokument: http://opensource.mit.edu/papers/garzarelli.pdf.

Gates, B., 1976: An Open Letter to Hobbyists. Als Online Dokument: http://www.flora.ca/gates_letter.txt.

Gehlen, A., 1957: Die Seele im technischen Zeitalter. Hamburg: Rowohlt.

Ghosh, R. A./Prakash, V.V., 2000: The Orbiten Free Software Survey. Als Online-Dokument: http://www.firstmonday.dk/issues/issue5_7/ghosh/.

Ghosh, R.A./Robles, G./Glott, R., 2002: Free/Libre and Open Source Software: Survey and Study. Deliverable D18: Final Report. Part V: Software Source Code Survey. Als Online-Dokument: http://www.infonomics.nl/FLOSS/report/FLOSS_Final5all.pdf.

Giddens, A., 1988: Die Konstitution der Gesellschaft. Frankfurt: Suhrkamp.

Granovetter, M.S., 1985: Economic Action and Social Structure. The Problem of Embeddedness. In: American Journal of Sociology 91.

Grassmuck, V., 2002: Freie Software. Zwischen Privat- und Gemeineigentum. Bonn: Bundeszentrale für politische Bildung.

Green, D.P./Shapiro, I., 1999: Rational Choice. Eine Kritik am Beispiel von Anwendungen in der Politischen Wissenschaft. München: Oldenbourg.

Greve, G. 2003: Brave GNU-World. Die monatliche GNU-Kolumne. In: Linux-Magazin 11/2003. Als Online-Dokument: http://www.linux-magazin.de/Artikel/ausgabe/2003/11/gnu/gnu.html.

Groß, M., 2003: Inventing Nature. Ecological Restoration by Public Experiments. Lanham u.a.O.: Lexington Books.

Groß, M./Hoffmann-Riem, H./Krohn, W., 2003: Realexperimente: Robustheit und Dynamik ökologischer Gestaltungen in der Wissensgesellschaft. In: Soziale Welt Vol. 54, Nr. 3.

Groß, M./Hoffmann-Riem, H./Krohn, W., 2005: Realexperimente. Ökologische Gestaltungsprozesse in der Wissensgesellschaft. Bielefeld: transcript.

Grün, C., 1979: Die zeitliche Schranke des Urheberrechts. Eine historische und dogmatische Erklärung. Bern: Stämpfli.

Grunwald, A., 2002: Technikfolgenabschätzung. Berlin: Sigma.

Habermas, J., 1973: Wahrheitstheorien. In: H. Fahrenbacher (Hrsg.): Wirklichkeit und Reflexion. Pfullingen: Neske.

Guala, F., 2003: Experimental Localism and External Validity. In: Philosophy of Science East Lansing. Vol. 70, Nr. 5.

Haberstumpf, H., 1996: Handbuch des Urheberrechts. Neuwied u.a.O.: Luchterhand.

Haberstumpf, H., 2000: Handbuch des Urheberrechts. 2. neubearb. und erw. Aufl. Neuwied u.a.O.: Luchterhand.

Hack, L., 1995: TA als theoriegeleitete Interventionsstrategie. Der Ansatz des „Constructive Technology Assessment/CTA“ in der sozialwissenschaftlichen Technikdebatte. Forschungszentrum Karlsruhe: Wissenschaftliche Berichte FZKA 5641.

Hacking, I., 1983: Representing and Inventing. Introductory Topics in the Philosophy of Natural Science. Cambridge u.a.O.: Cambridge University Press.

Hacking, I., 1996: Einführung in die Philosophie der Naturwissenschaften. Stuttgart: Reclam.

Hagstrom, W.O., 1965: The Scientific Community. New York: Basic Books.

Hagstrom, W.O., 1982: Gift as an organizating principle in science. In: B. Barnes/D. Edge (Hrsg.): Science in Context. Readings in the Sociology of Science. Cambridge: MIT Press.

Hardin, R., 1982: Collective Action. Baltimore; London: The John Hopkins University Press.

Harré, R., 2003: The Materiality of Instruments in a Metaphysics for Experiments. In: H. Radder (Hrsg.): The Philosophy of Scientific Experimentation. Pittburgh: University of Pittburgh Press.

Hasse, R./Krücken, G., 2005 [1999]: Neo-Institutionalismus. Bielefeld: transcript Verlag.

Healy, K./Schussman, A., 2003: The Ecology of Open-Source Software Development. Als Online-Dokument: http://opensource.mit.edu/papers/healyschussman.pdf.

Heidelberger, M., 1998: Die Erweiterung der Wirklichkeit im Experiment. In: M. Heidelberger/F. Steinle (Hrsg.): Experimental Essays – Versuche zum Experiment. Baden-Baden: Nomos.

Heinbokel, T. 1994: Benutzerbeteiligung: Schlüssel zum Erfolg oder Hemmschuh der Entwicklung? In: C. Frese/M. Brodbeck (Hrsg.): Produktivität und Qualität in Software-Projekten. Psychologische Analyse und Optimierung von Arbeitsprozessen in der Software-Entwicklung. München; Wien: Oldenbourg.

Heintz, B., 1993: Die Herrschaft der Regel. Zur Grundlagengeschichte des Computers. Frankfurt; New York: Campus.

Helms, H.G., 1983: Die Computer und die Phantasie. Auswirkungen der Mikroelektronik auf das Denken. In: Merkur 37/8.

Herbold, R., 2000: Technische Praxis und wissenschaftliche Erkenntnis. Norderstedt: Books on demand.

Herrlyn, G., 2002: Stabilisierende und Destabilisierende Wirkungen alltäglicher Technikerfahrungen im Spiegel biografischer Selbstdeutung. In: A. Epp/N.C. Taubert/A. Westermann (Hrsg.): Technik und Identität. Bielefeld: IWT-Paper 26.

Hertel, G./Niedner, S./Herrmann, S., 2003: Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. In: Research Policy 32.

Heymann, M./Wengenroth, U., 2001: Die Bedeutung von tacit knowledge bei der Gestaltung von Technik. In: U. Beck/W. Bonß, W. (Hrsg.): Die Modernisierung der Moderne. Frankfurt: Suhrkamp.

Hill, P.B., 2002: Rational-Choice Theorie. Bielefeld: transcript Verlag.

Himanen, P., 2001: The Hacker Ethic and the Spirit of the Information Age. London: Secker & Warburg.

Hippel, E.v., 2001: Open Source Shows the Way: Innovation by and for Users – No Manufacturer Required. Als Online-Dokument: http://opensource.mit.edu/papers/evhippel-osuserinnovation.pdf.

Hirsch-Kreinsen, H., 1997: Innovationsschwäche der deutschen Industrie. Wandel und Probleme von Innovationsprozessen. In: W. Rammert /G. Bechmann (Hrsg.): Technik und Gesellschaft. Jahrbuch 9. Frankfurt; New York: Campus.

Hirschmann, A.O., 1970: Exit, voice, and loyality: responses to decline in firms, organizations, and states. Cambridge: Harvard Univ. Press.

Hofmann, J., 1999: „Let a thousand proposals bloom“ – Mailing-Listen als Forschungsgegenstand. In: B. Batinic/A Werner/L. Gräf/W. Bandilla (Hrsg.): Online Research. Methoden, Anwendungen und Ergebnisse. Göttingen et al.: Hogrefe.

Horns, A.H., 2000: Der Patentschutz für softwarenbezogene Erfindungen im Verhältnis zu „Open-Source“-Software. In: JurPC Web-Dok. 223/2000, Abs. 1-80.

Hughes, T., 1985: Edison an electric Light. In: D. MacKenzie/J. Wajcman (Hrsg.): The Social Shaping of Technology. How the refrigerator got its hum. Philadelphia: Open University Press.

Hughes, T., 1991: Die Erfindung Amerikas. Der technologische Aufstieg der USA seit 1870. München: Beck.

Janich, P., 1998: Was macht experimentelle Resultate empiriehaltig? Die methodisch-kulturalistische Theorie des Experiments. In: M. Heidelberger/F. Steinle (Hrsg.): Experimental Essays –Versuche zum Experiment. Baden-Baden: Nomos.

Jansen, D., 2003: Einführung in die Netzwerkanalyse. 2. Aufl. Opladen: Leske + Budrich.

Joas, H., 1988: Symbolischer Interaktionismus. Von der Philosophie des Pragmatismus zu einer soziologischen Forschungstradition. In: Kölner Zeitschrift für Soziologie und Sozialpsychologie 40/2.

Joas, H., 1989: Praktische Intersubjektivität. Die Entwicklung des Werkes von George Herbert Mead. Frankfurt: Suhrkamp.

Johnson, F.G., 1980: Der Computer und die Technologie des Inneren. In: Psyche XXXIV.

Jones, S.G., 1995: Understanding the Community in the Information Age. S.G. Jones (Hrsg.): Cybersociety. Computer-Mediated Communication and Community. Thousand Oaks u.a.O.: Sage.

Jones, S.G., 1996: Using the News: An Examination of the Value and Use of News Sources in CMC. In: Journal of Computer-Mediated Communication 2/4.

Kappelhoff, P., 2000: Der Netzwerkansatz als konzeptioneller Rahmen für eine Theorie interorganisationaler Netzwerke. In: J. Sydow/A.

Windeler (Hrsg.): Steuerung von Netzwerken. Konzepte und Praktiken. Opladen: Westdeutscher Verlag.

Knie, A., 1989: Das Konservative des technischen Fortschritts. Zur Bedeutung von Konstruktionstraditionen, Forschungs- und Konstruktionsstilen in der Technikgenese. WZB: Discussion Paper.

Knie, A., 1994: Gemachte Technik. Zur Bedeutung von „Fahnenträgern“, „Promotoren“ und „Definitionsmacht“ in der Technikgenese. In: W. Rammert/G. Bechmann (Hrsg.): Technik und Gesellschaft. Jahrbuch 7. Frankfurt; New York: Campus.

Knorr-Cetina, K., 1984: Die Fabrikation von Erkenntnis: Zur Anthropologie der Naturwissenschaft. Frankfurt: Suhrkamp.

Köberle, S./Gloeden, F./Hennen, L., 1997: Einleitung. In: S. Köberle/F. Gloeden/L. Hennen (Hrsg.): Diskursive Verständigung. Mediation und Partizipation in Technikkontroversen. Baden-Baden: Nomos.

Koch, F.A./Schnupp, P., 1991: Software-Recht. Bd. 1. Berlin u.a.O.: Springer.

Koch, S./Schneider, G., 2002: Effort, co-operation and co-ordination in an open source software project: GNOME. In: Information Systems Journal 12/1.

Konrad, W./Paul, G., 1999: Innovationen in der Softwareindustrie. Organisation und Entwicklungsarbeit. Frankfurt; New York: Campus.

Kowol, U., 1996: Innovationsnetzwerke – Rekursive Technikentwicklung zwischen Nutungsvisionen und Verwendungspraxis. Bielefeld: Dissertation.

Kowol, U., 1998: Innovationsnetzwerke – Rekursive Technikentwicklung zwischen Nutzungsvisionen und Verwendungspraxis. Wiesbaden: Deutscher Universitäts-Verlag.

Kowol, U./Krohn, W., 1997: Modernisierungsdynamik und Innovationslethargie: Auswege aus der Modernisierungsklemme. In: B. Blättel-Mink (Hrsg.): Zwischen Akteur und System. Die Organisation der Innovation. Opladen: Westdeutscher Verlag.

Kowol, U./Krohn, W., 2000: Innovation und Vernetzung – die Konzeption der Innovationsnetzwerke. In: J. Weyer (Hrsg.): Soziale Netzwerke. München; Wien: Oldenbourg.

Krämer, S., 1988: Symbolische Maschinen. Die Idee der Formalisierung in geschichtlichem Abriß. Darmstadt: Wiss. Buchgesellschaft.

Krohn, W., 1997a: Die Innovationschancen partizipatorischer Technikgestaltung und diskursiver Konfliktregulierung. In: S. Köberle/F. Gloede/L. Hennen (Hrsg.): Diskursive Verständigung? Mediation und Partizipation in Technikkontroversen. Baden-Baden: Nomos.

Krohn, W., 1997b: Rekursive Lernprozesse: Experimentelle Praktiken in der Gesellschaft. Das Beispiel der Abfallwirtschaft. In: W. Rammert

(Hrsg.): Jahrbuch für Technik und Gesellschaft. Frankfurt; New York: Campus.

Krohn, W., 2003: Zukunftsgestaltung als Experimente von und mit uns selbst. Wer kann das verantworten? In: K. Mensch/J.C. Schmidt (Hrsg.): Technik und Demokratie. Zwischen Expertokratie. Parlament und Bürgerbeteiligung. Opladen: Leske + Budrich.

Krohn, W./van den Daele, W., 1998: Science as an Agent of Change: Finalization and Experimental Implementation. In: Social Science Information 37/1.

Krohn, W./Weingart, P., 1986: ‚Tschernobyl' – das größte anzunehmende Experiment. In: Kursbuch 85.

Krohn, W./Weingart, P., 1987: Commentary: Nuclear Power as a Social Experiment – European Political „Fall Out" from the Chernobyl Meltdown. In: Science, Technology & Human Values 12.

Krohn, W./Weyer, J., 1989: Gesellschaft als Labor. Die Erzeugung sozialer Risiken durch experimentelle Forschung. In: Soziale Welt 40/3.

Krohn, W./Weyer, J., 1990: Gesellschaft als Labor. Risikotransformation und Risikokonstitution durch moderne Forschung. In: J. Halfmann/K.P. Japp (Hrsg.): Riskante Entscheidungen und Katastrophenpotentiale: Elemente einer soziologischen Risikoforschung. Opladen: Westdeutscher Verlag.

Krücken, G., 1994: Risikosoziologie. Stand und Perspektiven der sozialwissenschaftlichen Risikoforschung. In: G. Bechmann (Hrsg.): Technik und Gesellschaft. Jahrbuch 7. Frankfurt; New York: Campus.

Krücken, G., 2002: Amerikanischer Neo-Institutionalismus – europäische Perspektiven. In: Sociologia Internationalis 40.

Kuhn, T., 1973: Die Struktur wissenschaftlicher Revolutionen. Frankfurt: Suhrkamp.

Lakhani, K.R./Wolf, R.G., 2003: Why Hackers Do What They Do: Understanding Motivation Effort in Free/Open Source Software Projects. MIT Sloan School of Management. Working Paper 4425-03.

Latour, B./Wolgar, S., 1982: The cycle of credibility. In: B. Barnes/D. Edge (Hrsg.): Science in Context. Readings in the Sociology of Science. Cambridge: MIT Press.

Lehmann, M., 1996: TRIPS/WTO und der internationale Schutz von Computerprogrammen. In: Computer und Recht 1/1996.

Leinemann, F., 1998: Die Sozialbindung des Geistigen Eigentums. Zu den Grundlagen der Schranken des Urheberrechts zugunsten der Allgemeinheit. Baden-Baden: Nomos.

Lerner, J./Tirole J., 2000: The Simple Economics Of Open Source. Working Paper 7600. Cambridge: National Bureau of Economic Research.

Lerner, J./Tirole, J., 2001: The open source movement: Key research questions. In: European Economic Review 45.

Lerner, J./Tirole, J., 2002: Some Simple Economics Of Open Source. The Journal of Industrial Economics 50/2.

Lerner, J./Tirole, J., 2002a: The Scope of Open Source Licenses. MIT Working Paper.

Lewinsky, S.v., 1997: Die WIPO-Verträge zum Urheberrecht und zu verwandten Schutzrechten vom Dezember 1996. In: Computer und Recht 7/1997.

Llewelyn, A.I./Wickens, R.F., 1969: The Testing of Computer Software. In: P. Naur/B. Randell (Hrsg.): Software-Engineering. Report on a conference sponsored by the NATO Science Committee. Garmisch, 7th to 11th October 1968.

Luhmann, N., 1964: Funktion und Folgen formaler Organisation. Berlin: Duncker & Humblot.

Luhmann, N., 1970: Soziologische Aufklärung 1. Opladen: Westdeutscher Verlag.

Luhmann, N., 1984: Soziale Systeme. Grundriß einer allgemeinen Theorie. Frankfurt: Suhrkamp.

Luhmann, N., 1986: Ökologische Kommunikation. Opladen: Westdeutscher Verlag.

Luhmann, N., 1990: Die Wissenschaft der Gesellschaft. Frankfurt: Suhrkamp.

Luhmann, N., 1997: Die Gesellschaft der Gesellschaft. 2 Bde. Frankfurt: Suhrkamp.

Luhmann, N., 2001: Zuversicht, Vertrauen. Probleme und Alternativen. In: M. Hartmann/C. Offe (Hrsg.): Vertrauen. Die Grundlage des sozialen Zusammenhalts. Frankfurt; New York: Campus.

Lutterbeck, B./Gehring, R./Horns, A.H., 2000: Sicherheit in der Informationstechnologie und Patentschutz für Software-Produkte – ein Widerspruch? Kurzgutachten im Auftrag des Bundesministeriums für Wirtschaft und Technologie. TU Berlin.

MacKenzie, D.A./Wajcman, J. (Hrsg.) 1987 : The social shaping of technology : how the refrigerator got its hum. Reprint. Milton Keynes et al.: Open University Press.

MacKenzie, D.A./Wajcman, J. (Hrsg.) 1999: The Social Shaping of Technology. Women’s Technobiographies. 2. Aufl. Buckingham: Open University Press.

MacKenzie, D.A./Wajcman, J., 1985: Introductory essay: The social shaping of technology. In: D. MacKenzie/J. Wajcman (Hrsg.): The Social Shaping of Technology. How the refrigerator got its hum. Philadelphia: Open University Press.

Madanmohan, T.R./Navelkar, S., 2002: Roles and Knowledge Management in Online Technology Communities: An Ethnography Study. Als Online-Dokument: http://opensource.mit.edu/papers/madanmohan2.pdf.

March, J.G., 1990: Beschränkte Rationalität. Ungewißheit und die Technik der Auswahl. In: J.G. March (Hrsg.): Entscheidung und Organisation. Kritische und konstruktive Beiträge, Entwicklungen und Perspektiven. Wiesbaden: Gabler.

March, J.G./Simon, H.A., 1958: Organizations. New York et al.: Wiley.

March, J.G./Simon, H.A., 1977: Kognitive Grenzen der Rationalität. In. E. Witte/A.L. Thimm, A.L. (Hrsg.): Entscheidungstheorie. Texte und Analysen. Wiesbaden: Gabler.

Marcuse, H., 1967: Der eindimensionale Mensch. Studien zur Ideologie der fortgeschrittenen Industriegesellschaft. Neuwied: Luchterhand.

Marx, K., 1969: Das Kapital. Kritik der politischen Ökonomie. Bd. 1. MEW 23. Berlin: Dietz.

Mauss, M., 1954: The Gift. Forms and Functions of Exchange in Archaic Societies. Transl. by Ian Cunnison. London: Cohen & West.

Mauss, M., 1975: Soziologie und Anthropologie. Bd. 2. Gabentausch – Soziologie und Psychologie – Todesvorstellung – Körpertechniken – Begriff der Person. München; Wien: Hanser.

Mayntz, R., 1992: Modernisierung und die Logik von interorganisatorischen Netzwerken. In: Journal für Sozialforschung 32/1.

Mayring, R. 1995: Qualitative Inhaltsanalyse. In: U. Flick/E. v. Kardorff/H. Keupp/L. v. Rosenstil/S. Wolff (Hrsg.): Handbuch qualitative Sozialforschung. 2. Aufl. Weinheim: Beltz.

McCracken, D.D., 1973: Revolution in Programming: An Overview. In: Datamation 19/12.

McKusick, M., 1999: Twenty Years of Berkeley Unix. From AT&T-Owned to Freely Redistributable. In: C. DiBona/S. Ockman/M. Stone (Hrsg.): Open Sources. Voices from the Open Source Revolution. Bejing et al.: O’Reilly.

McSummit, B./Martin, J., 1989: Die Silicon Valley Story. Konstanz: Artigas.

Mead, G.H., 1973: Geist, Identität und Gesellschaft: aus der Sicht des Sozialbehaviorismus. Frankfurt: Suhrkamp.

Meretz, S., 1999: Linux – Software-Guerilla oder mehr? Die Linux-Story als Beispiel für eine gesellschaftliche Alternative. In: FifF Kommunikation Jg. 99, Nr. 3.

Meretz, S., 2000: Linux & Co. Freie Software. Ideen für eine andere Gesellschaft. Neu-Ulm: AG Spak Bücher.

Merton, R.K., 1938: Science and the Social Order. In: R.K. Merton (Hrsg.) 1973: The Sociology of Science. Theoretical and empirical Investigations. Chicago; London: The University of Chicago Press.

Merton, R.K., 1942: The normative Structure of Science. In: R.K. Merton (Hrsg.), 1973: The Sociology of Science. Theoretical and empirical Investigations. Chicago; London: The University of Chicago Press.

Merton, R.K., 1957: Priorities in Scientific Discovery. In: R.K. Merton (Hrsg.), 1973: The Sociology of Science. Theoretical and empirical Investigations. Chicago; London: The University of Chicago Press.

Merton, R.K., 1968: The Matthew Effect in Science. The reward and communication systems of science are considered. In: Science 159 (3810).

Merton, R.K., 1972: Wissenschaft und demokratische Sozialstruktur. In: P. Weingart (Hrsg.): Wissenschaftssoziologie. Frankfurt: Fischer.

Merton, R.K., 1988: The Matthew Effect in Science II. Cumulative Advantage and the Symbolism of Intellectual Property. In: ISIS.

Meyer, J.W./Rowan, B., 1991: Institutionalized Organizations: Formal Structure as Myth and Ceremony. In: P.J. DiMaggio/W.W. Powell (Hrsg.): The New Institutionalism in Organizational Analysis. Chicago; London: The University of Chicago Press.

Miller, F.E./Lindamood, G.E., 1973: Structured Programming: Top-down Approach. In: Datamation 19/12.

Miller, M., 1992: Rationaler Dissens. Zur gesellschaftlichen Funktion sozialer Konflikte. In: H.-J. Giegel (Hrsg.): Kommunikation und Konsens in modernen Gesellschaften. Frankfurt: Suhrkamp.

Mitra, A., 1999: Characteristics of the WWW Text: Tracing Discriptive Strategies. In: Journal of Computer-Mediated Communication 5.

Moody, G., 2001: Die Software Rebellen. Die Erfolgsstory von Linus Torvalds und Linux. Landsberg: Verlag moderne Industrie.

Moritz, H.-W., 1986: Computersoftware: Rechtsschutz und Vertragsgestaltung. München: Beck.

Moritz, H.-W., 1993a: Softwarelizenzverträge (I). In: Computer und Recht 5/1993.

Moritz, H.-W., 1993b: Softwarelizenzverträge (II). In: Computer und Recht 6/1993.

Mozetič, G., 1998: Wieviel muß die Soziologie über Handlungen wissen? In: A. Balog/M. Gabriel (Hrsg.): Soziologische Handlungstheorie. Österreichische Zeitschrift für Soziologie, Sonderband 4. Opladen: Westdeutscher Verlag.

Mumford, L., 1934: Technics and Civilization. New York: Harcourt.

Myers, G.J., 1979: The Art of Software Testing. New York u.a.O.: John Wiley & Sons.

Nagl, M., 1990: Softwaretechnik. Methodisches Programmieren im Großen. Berlin u.a.O.: Springer.

Narduzzo, A./Rossi, A., 2003: Modularity in Action: GNU/Linux and Free/Open Source Software Development Moedel Unleashed. Als Online-Dokument: http://opensource.mit.edu/papers/narduzzorossi.pdf.

Naur, P./Randell, B. (Hrsg.), 1969: Software-Engineering. Report on a conference sponsored by the NATO Science Committee. Garmisch, 7th to 11th October 1968.

Nordemann, W., 2002: Das neue Urhebervertragsrecht. München: Beck.

O'Mahony, S., 2003: Guarding the commons: how community managed software projects protect their work. In: Research Policy 32.

Ogburn, W.F., 1966: Social Change. With Respect to Cultural and Original Nature. New York: Delta Book.

Olson, M., 1968: Die Logik des kollektiven Handelns. Kollektivgüter und die Theorie der Gruppe. Tübingen: Mohr.

Osterloh, M./Rota, S./Kuster, B., o.J.: Open Source Software Production: Climbing on the Shoulders of Giants. Als Online-Dokument: http://opensource.mit.edu/papers/osterlohrotakuster.pdf.

Ostrom, E., 1999: Die Verfassung der Allmende. Tübingen: Mohr.

Paccagnella, L., 1997: Getting the Seats of Your Pants Dirty: Strategies for Ethnographic Research on Virtual Communities. In: Journal of Computer-Mediated Communication 3/1.

Pagel, B.V./Six, H.-W., 1994: Software Engineering. Band 1: Die Phasen der Softwareentwicklung. Bonn u.a.O.: Addison-Wesley.

Pahud, E., 2000: Die Sozialbindung des Urheberrechts. Universität Zürich: Dissertation.

Parnas, D.L., 1972: On the Criteria To Be Used in Decomposing Systems into Modules. In: Communications of the ACM 15/12.

Paschen, H./Petermann, T., 1991: Technikfolgen-Abschätzung – Ein strategisches Rahmenkonzept für die Analyse und Bewertung von Technik. In: T. Petermann (Hrsg.): Technikfolgen-Abschätzung als Technikforschung und Politikberatung. Frankfurt u.a.O.: Campus.

Perrow, C., 1984: Normal Accidents: Living with High-Risk Technologies. New York: Basic Books.

Peuchert, E./Scheer, G., 2003: Soziologische Theorie. In: B. Schäfers (Hrsg.): Grundbegriffe der Soziologie. Opladen: Leske + Budrich.

Pinch, T., 1998: Theoretical Approaches to Science, Technology and Social Change: Recent Developments. In: Southeast Asian Journal of Social Science 26/1.

Pinch, T./Bijker, W.E., 1984: The social construction of facts and artefacts: or How the Sociology of Science and the Sociology of Technology might Benefit Each Other. In: Social Studies of Science 14/1.

Pinch, T./Bijker, W.E., 1986: Science, Relativism and the New Sociology of Technology: Reply to Russel. In: Social Studies of Science 16/2.

Pomberger, G., 1993: Grundlagen des Software-Engineering: Prototyping und objektorientierte Software-Entwicklung. München; Wien: Hanser.

Pommerehne, W. W., 1987: Präferenzen für öffentliche Güter : Ansätze zu ihrer Erfassung. Tübingen: Mohr.

Popper, K.R., 1979: Die beiden Grundprobleme der Erkenntnistheorie. Tübingen: Mohr.

Popper, K.R., 1982: Logik der Forschung. 7. Aufl. Tübingen: Mohr.

Powell, W.W., 1990: Neither Market nor Hierarchy. Network Forms of Organization. In: Research in Organizational Behavior 12.

Powell, W.W., 1996: Weder Markt noch Hierarchie: Netzwerkartige Organisationsformen. In: P. Kenis/V. Schneider (Hrsg.): Organisation und Netzwerk. Institutionelle Steuerung in Wirtschaft und Politik. Frankfurt; New York: Campus.

Preisendörfer, P., 1995: Vertrauen als soziologische Kategorie. Möglichkeiten und Grenzen einer entscheidungstheoretischen Fundierung des Vertrauenskonzepts. In: Zeitschrift für Soziologie 24/4.

Pres, A., 1994a: Gestaltungsformen urheberrechtlicher Softwarelizenzverträge: eine juristische und ökonomische Untersuchung unter besonderer Berücksichtigung des Zweiten Gesetzes zur Änderung des Urheberrechtsgesetzes vom 9. Juni 1993. Köln: Schmidt.

Pres, A., 1994b: Gestaltungsformen urheberrechtlicher Softwarelizenzverträge. In: Computer und Recht 9/1994.

Purdy, G.N., 2000: CVS kurz & gut. Bejing u.a.O.: O'Reilly.

Radder, H., 2003a: Towards a More Developed Philosophy of Scientific Experimentation. In: H. Radder (Hrsg.): The Philosophy of Scientific Experimentation. Pittburgh: University of Pittburgh Press.

Radder, H., 2003b: Technology and Theory in Experimental Science. In: H. Radder (Hrsg.): The Philosophy of Scientific Experimentation. Pittburgh: University of Pittburgh Press.

Rammert, W., 1991: Vom Umgang mit Computern im Alltag. Fallstudien zur Kultivierung einer neuen Technik. Opladen: Westdeutscher Verlag.

Rammert, W., 1993: Technik aus soziologischer Perspektive. Opladen: Westdeutscher Verlag.

Rammert, W., 1994: Vom Nutzen der Technikgeneseforschung für die Technikfolgenabschätzung. In: G. Bechmann/T. Petermann (Hrsg.): Interdisziplinäre Technikforschung: Genese, Folgen, Diskurs. Frankfurt: Campus.

Rammert, W., 1995: Computerwelten: Vollendung der Moderne oder Epochenbruch zur Postmoderne? Soziologische Revue 18.

Rammert, W., 2000: Technik aus soziologischer Perspektive 2. Opladen: Westdeutscher Verlag.

Randell, B., 1996: NATO Software Engineering Reports. In: W. Aspray/R. Keil-Slawic/D.L. Parnas (Hrsg.): History of Software Engineering. Position Papers for Dagstuhl Seminar 9635.

Raubenheimer, A., 1994: Softwareschutz nach dem neuen Urheberrecht. In: Computer und Recht 2/1994.

Raymond, E.S. (Hrsg.) 1998: The New Hacker's Dictionary. 3rd Ed. London: MIT-Press.

Raymond, E.S. 1999: The Cathedral and the Bazaar. Musings on Linux and Open Source By an Accidental Revolutionary. Bejing u.a.O.: O'Reilly.

Raymond, E.S., 2001: How To Become A Hacker. Version 1.7. Als Online-Dokument: http://www.tuxedo.org/~esr/faqs/hacker-howto.html

Raymond, E.S., 2003: Basics of the Unix Philosophy. Als Online-Dokument: http://www.faqs.org/docs/artu/ch01s06.html#id2877537.

Rehbinder, M., 1998: Urheberrecht. Ein Studienbuch. 10., neubearb. Aufl. München: Beck.

Rehbinder, M., 2002: Urheberrecht. 12. Aufl. München: C.H. Beck.

Rheingold, H., 1992: Der Alltag in meiner virtuellen Gemeinschaft. In: M. Faßler/W.R. Halbach (Hrsg.): Cyberspace. Gemeinschaften, Virtuelle Kolonien, Öffentlichkeit. München: Fink.

Rheingold, H., 1994: Virtuelle Gemeinschaft. Soziale Beziehungen im Zeitalter des Computers. Bonn u.a.O.: Addison-Wesley.

Rigamonti, C.P., 2001: Geistiges Eigentum als Begriff und Theorie des Urheberrechts. Baden-Baden: Nomos.

Rode, H./Hansen, H.K., 1993: Die Erfindung der universellen Maschine: Zur Geschichte der Datenverarbeitung in 50 Jahrhunderten. Hannover: Metzler.

Rogers, E.M./Larsen J.K., 1986: Silicon Valley Fieber. An der Schwelle zur High-Tech-Zivilisation. Berlin: Diogenes.

Rölle, D./Blättel-Mink, B., 1998: Netzwerke in der Organisationssoziologie – neuer Schlauch für alten Wein? In: Österreichische Zeitschrift für Soziologie 23/3.

Russel, S., 1986: The Social Construction Artefacts. A Response to Pinch and Bijker. In: Social Studies of Science 16.

Scacchi, W., 2003: Free/Open Source Software Development Practices in the Computer Game Community. Als Online-Dokument: http://www.ics.uci.edu/%7Ewscacchi/Papers/New/FOSS-Development Practices.pdf.

Schack, H., 1997: Urheber- und Urhebervertragsrecht. Tübingen: Mohr.

Schack, H., 1998: Neue Techniken und Geistiges Eigentum. In: Juristenzeitung 15/16.

Schack, H., 2001: Urheber- und Urhebervertragsrecht. 2. neubearb. Aufl. Tübingen: Mohr.

Schelsky, H., 1965: Der Mensch in der wissenschaftlichen Zivilisation. In: H. Schelsky: Auf der Suche nach Wirklichkeit. Düsseldorf: Diederich.

Schilcher, T., 1989: Der Schutz des Urhebers gegen Werksänderungen. München: Florentz.

Schimank, U., 1996: Theorien gesellschaftlicher Differenzierung. Opladen: Leske + Budrich.

Schmied, G., 1996: Schenken. Über eine Form sozialen Handelns. Opladen: Leske + Budrich.

Schmookler, J., 1966: Invention and Economic Growth. Cambridge, Massachusetts: Harvard University Press.

Schneider, H.-J., 1998: Lexikon Informatik und Datenverarbeitung. 4. Aufl. München; Wien: Oldenbourg.

Schoder, D./Fischbach, K., 2002: Peer-to-Peer. Anwendungsbereiche und Herausforderungen. In: D. Schoder/K. Fischbach/R. Teichmann (Hrsg.): Peer-to-Peer. Ökonomische, technische und juristische Perspektiven. Heidelberg: Springer.

Schulz-Schaeffer, I., 1996: Software-Entwicklung zwischen Ingenieur- und Designwissenschaft. Überzeugungskraft und nützliche Widersprüchlichkeit von Software-Engineering und Software-Gestaltung. In: H.D. Hellige (Hrsg.): Leitbilder auf dem Prüfstand. Leitbild-Assessment aus Sicht der Informatik und Computergeschichte. Berlin: Sigma.

Schulz-Schaeffer, I., 2002: Sozialtheorie der Technik. Frankfurt; New York: Campus.

Schumpter, J.A., 1950: Kapitalismus, Sozialismus und Demokratie. 2. Aufl. Bern: A. Francke.

Scott, W.R., 1986: Grundlagen der Organisationstheorie. Frankfurt; New York: Campus.

Scott, W.R., 1995: Institutions and Organizations. Thousand Oaks u.a.O.: Sage.

Selig, F., 1969: Figure 2. From Selig: Documentation for service and users. In: P. Naur/B. Randell (Hrsg.): Software-Engineering. Report on a conference sponsored by the NATO Science Committee. Garmisch, 7th to 11th October 1968.

Sen, A., 1986: Behaviour and the Concept of Preference. In: J. Elster (Hrsg.): Rational choice. Readings in social and political theory. Oxford: Blackwell.

Sieber, A./Henninger, A., 2001: Vorgehensmodelle, Projekttypen und Arbeitspraktiken als Bausteine zur realitätsnahen Beschreiung von Softwareentwicklung. In: W. Dilger/E. Keitel (Hrsg.): Kultur und Stil in der Informatik? Chemnitzer Informatik-Berichte, CSR-01-01.

Siekmann, J., 2000: Bravehack. Technische, wirtschaftliche und gesellschaftliche Aspekte von freier Software und Open Source; ihr Wesen, ihre Geschichte, ihre Organisation und Projekte. Version 1.0. Als Online-Dokument: http://www.bravehack.de.

Silverman, D., 1970: Theorie der Organisationen. Soziologische Aspekte zu System, Bürokratie und Management. Wien u.a.O.: Böhlau.

Simon, H.A., 1957: Models of Man. Mathematical Essays on Rational Human Behavior in a Social Setting. New York: John Wiley.

Sneed, H.M., 1986: Software Entwicklungsmethodik. 5. überarb. Aufl. Köln: Müller.

Sonnentag, S., 1994: Spitzenkräfte: Fachwissen allein genügt nicht. In: C. Frese/M. Brodbeck (Hrsg.): Produktivität und Qualität in Software-Projekten. Psychologische Analyse und Optimierung von Arbeitsprozessen in der Software-Entwicklung. München; Wien: Oldenbourg.

Sorge, A., 1985: Informationstechnik und Arbeit im sozialen Prozeß. Arbeitsorganisation, Qualifikation und Produktivkraftentwicklung. Frankfurt; New York: Campus.

Stach, H., 1998: Beschreiben, konstruieren, programmieren. Zur Verschmelzung von Theorie und Gegenstand. In: D. Siefkes/P. Eulenhöfer/H. Stach/H. Städtler (Hrsg.): Sozialgeschichte der Informatik. Kulturelle Praktiken und Orientierungen. Wiesbaden: Deutscher Universitäts-Verlag.

Stach, H., 2001: Zwischen Organismus und Notation. Zur kulturellen Konstruktion des Computer-Programms. Wiesbaden: Deutscher Universitäts-Verlag.

Stallman, R.S., 1983: Initial Announcement. Veröffentlicht auf der Newsgroup net.unix-wizards,net.usoft am 27. Sept. 1983. Als Online-Dokument: http://www.gnu.org/gnu/initial-announcement.html.

Stallman, R.S., 1985: The GNU Manifesto. Als Online-Dokument: http://www.gnu.org/gnu/manifesto.html.

Stallman, R.S., 1992: Why Software Should Be Free. Als Online-Dokument: http://www.gnu.org/philosophy/shouldbefree.html.

Stallman, R.S., 1994: Why Software Should Not Have Owners. Als Online-Dokument: http://www.gnu.org/philosophy/why-free.html.

Stallman, R.S., 1999: The GNU Operating System and the Free Software Movement. In: C. DiBona/S. Ockman/M. Stone (Hrsg.): Open Source. Voices from the Open Sopurce Revolution. Bejing u.a.O.: O'Reilly.

Stallman, R.S., 2000: Freedom – Or Copyright? Als Online-Dokument: http://www.gnu.org/philosophy/freedom-or-copyright.html.

Stallman, R.S., 2002: Free Software Free Society. Selected essays of Richard M. Stallman. Boston: GNU Press.

Steinle, F., 1998: Exploratives vs. theoriebestimmtes Experimentieren: Ampères erste Arbeiten zum Elektromagnetismus. In: M. Heidelberger/F. Steinle, F. (Hrsg.): Experimental Essays – Versuche zum Experiment. Baden-Baden: Nomos.

Storer, N.W., 1972: Das soziale System der Wissenschaft. In: P. Weingart (Hrsg.): Wissenschaftssoziologie I. Frankfurt: Fischer.

Strasser, H./Vosswinkel, S., 1997: Vertrauen im gesellschaftlichen Wandel. In: M. Schweer (Hrsg.): Interpersonales Vertrauen: Theorien und empirische Befunde. Opladen: Westdeutscher Verlag.

Suhr, R./Suhr, R., 1993: Software Engineering. Technik und Methodik. München; Wien: Oldenbourg Verlag.

Sydow, J., 1992: Strategische Netzwerke. Evolution und Organisation. Wiesbaden: Gabler.

Sydow, J./Windeler, A., 2000: Steuerung von und in Netzwerken – Perspektiven, Konzepte, vor allem aber offene Fragen. In: J. Sydow/A. Windeler (Hrsg.): Steuerung von Netzwerken. Konzepte und Praktiken. Opladen: Westdeutscher Verlag.

Sztompka, P., 1999: Trust. A sociological Theory. Cambridge: University Press.

Teubner, G., 1992: Die vielköpfige Hydra. Netzwerke als kollektive Akteure höherer Ordnung. In: W. Krohn/G. Küppers (Hrsg.): Emergenz: Die Entstehung von Ordnung, Organisation und Bedeutung. Frankfurt: Suhrkamp.

Theißling, F., 1995: Auf dem Weg in die Softwarekrise? Computeranwendungen und Programmentwicklung in den USA der fünfziger

und sechziger Jahre. TU Berlin: Forschungsberichte des Fachbereichs Informatik. Bericht 95-14.

Torvalds, L., 1999: The Linux Edge. In: C. DiBona/S. Ockman/M. Stone (Hrsg.): Open Source. Voices from the Open Source Revolution. Bejing et al.: O'Reilly.

Torvalds, L./Diamond, D., 2001: Just For Fun. Wie ein Freak die Computerwelt revolutionierte. München; Wien: Hanser.

Tzouris, M., 2002: Software Freedom, Open Software and the Participant's Motivation. A Multidisciplinary Study. The London School of Economics and Political Science: Summer Dissertation.

Ulmer, E., 1980: Urheber- und Verlagsrecht. 3. neubearb. Aufl. Berlin u.a.O.: Springer.

Vatter, A., 1998: Politik. In: D. Rehmann-Sutter/A. Vatter/H. Seiler (Hrsg.): Partizipative Risikopolitik. Opladen: Westdeutscher Verlag.

Volpert, W., 1985: Zauberlehrlinge. Die Gefährliche Liebe zum Computer. Weinheim; Basel: Beltz.

Weber, A., 1980: Die Rationalitätenfalle in der Kollektivgüterökonomie. Sozioökonomische, finanzwissenschaftliche und ideengeschichtliche Bausteine zu einer Theorie der Kollektivgüter. Universität zu Köln: Dissertation.

Weber, H., 1992: Die Software-Krise und ihre Macher. Berlin u.a.O.: Springer.

Weingart, P., 1983: Verwissenschaftlichung der Gesellschaft, Politisierung der Wissenschaft. In: Zeitschrift für Soziologie 12/3.

Weingart, P., 1999: Paradoxes. Scientific expertise and political accountability: paradoxes of science in politics. In: Science and Public Policy 26/3.

Weingart, P., 2001: Die Stunde der Wahrheit? Zum Verhältnis der Wissenschaft zu Politik, Wirtschaft und Medien in der Wissensgesellschaft. Weilerswist: Velbrück Wissenschaft.

Weizenbaum, J., 1977: Die Macht der Computer und die Ohnmacht der Vernunft. Frankfurt: Suhrkamp.

Wellman, B., 2000: Die elektronische Gruppe als soziales Netzwerke. In: U. Thiedeke (Hrsg.): Virtuelle Gruppen. Charakteristika und Problemdimensionen. Opladen: Westdeutscher Verlag.

Weltz, F./Ortmann, R.G., 1992: Das Softwareprojekt. Projektmanagement in der Praxis. Frankfurt; New York: Campus.

Weyer, J., 1993: System und Akteur. Zum Nutzen zweier soziologischer Paradigmen bei der Erklärung erfolgreichen Scheiterns. In: Kölner Zeitschrift für Soziologie und Sozialpsychologie 45/1.

Weyer, J., 2000a: Einleitung. Zum Stand der Netzwerkforschung in den Sozialwissenschaften. In: J. Weyer (Hrsg.): Soziale Netzwerke. München; Wien: Oldenbourg Verlag.

Weyer, J., 2000b: Soziale Netzwerke als Mikro-Makro-Scharnier. Fragen an die soziologische Theorie. In: J. Weyer (Hrsg.): Soziale Netzwerke. München; Wien: Oldenbourg.

Weyer, J./Kirchner, U./Riedl, L./Schmidt, J.F.K., 1997: Technik, die Gesellschaft schafft. Soziale Netzwerke als Ort der Technikgenese. Berlin: Sigma.

Wichmann, T., 2002: Free/Libre Open Source Software Survey and Study. Use of Open Source Software in Firms and Public Institutions. Evidence from Germany, Sweden and UK. Final Report. Part 1.

Wienhöfer, H., 1996: Bürgerforen als Methoden einer partizipativen Technikfolgenbewertung. In: H. Wienhöfer (Hrsg.): Bürgerforen als Verfahren der Technikfolgenbewertung. Akademie für Technikfolgenabschätzung in Baden-Württemberg: Arbeitsbericht Nr. 67.

Wiesenthal, H., 2000: Markt, Organisation und Gemeinschaft als zweitbeste Verfahren sozialer Koordination. In: R. Werle/U. Schimank (Hrsg.): Gesellschaftliche Komplexität und kollektive Handlungsfähigkeit. Frankfurt; New York: Campus.

Williamson, O.E., 1981: The Economics of Organization: The Transaction Cost Approach. In: American Journal of Sociology 87/3.

Williamson, O.E., 1985: Then Economic Insitutions of Capitalism: Firms, Markets, Relational Contracting. New York: Free Press.

Williamson, O.E., 1986: Economic Organizations: Firms, Markets and Policy Control. Repr. New York: Harvester.

Williamson, O.E., 1996: Vergleichende ökonomische Organisationstheorie: Die Analyse diskreter Strukturalternativen. In: P. Kenis/V. Schneider (Hrsg.): Institutionelle Steuerung in Wirtschaft und Politik. Frankfurt; New York: Campus.

Willke, H., 2001: Systemtheorie III: Steuerungstheorie. 3. Aufl. Stuttgart: UTB.

Wilke, H., 1995: Systemtheorie III. Steuerungstheorie . Grundzüge einer Theorie der Steuerung komplexer Sozialsysteme. Frankfurt: Fischer.

Winner, L., 1993: Upon Opening the Black Box and Finding It Empty: Social Constructivism and the Philosophy of Technology. In: Science, Technology & Human Values 18/3.

Wittmann, R., 1991: Geschichte des Deutschen Buchhandels. München: C.H. Beck'sche Verlagsgesellschaft.

Wynne, B., 1982: Institutional Mythologies and Dual Societies in the Management of Risk. In: H.C. Kunreuther /E.V. Ley (Hrsg.): The

Risk Analysis Controversy. An Institutional Perspective. Berlin u.a.O.: Springer.

Zeitlyn, D., 2003: Gift economies in the development of open source software: anthropological reflections. In: Research Policy 32.

Zilahi-Szabó, M.G., 1995: Kleines Lexikon der Informatik und Wirtschaftsinformatik. München; Wien: Oldenbourg.

Zucker, L.G., 1991: The Role of Institutionalization in Cultural Persistence. In: P.J. DiMaggio/W.W. Powell (Hrsg.): The New Institutionalism in Organizational Analysis. Chicago; London: The University of Chicago Press.